

PATENT ABSTRACTS OF JAPAN

08-263447

11.10.1998

G06F 15/16
G06F 9/445
G06F 13/00

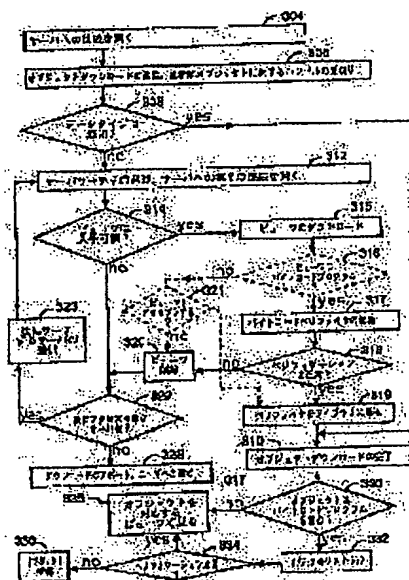
(71)Applicant : **SUN MICROSYST INC**

(72)Inventor : GOSLING JAMES A

Priority number : 94 359884 Priority date : 20.12.1994 Priority country : US

(57)Abstract:

SOLUTION: At the time of loading data (object) stored in another server or the like through a network and referring to them, the viewer corresponding to the object is automatically searched, and in the case that the appropriate viewer is found in the other server or the like, verification is performed so as to confirm the maintainability before activating the viewer. Especially, importance is placed in the verification relating to the use of a stack and a data type to the program written in a byte code language.



PF030016

correct

EP0718761

(11)特許出願公開番号

特開平8-263447

(43)公開日 平成8年(1996)10月11日

| (51) Int.Cl. ⁶ | | 識別記号 | 序内整理番号 | F I | 技術表示箇所 |
|---------------------------|-------|-------|---------|---------|---------|
| G 0 6 F | 15/16 | 3 7 0 | | G 0 6 F | 15/16 |
| | 9/445 | | 7368-5E | | 13/00 |
| | 13/00 | 3 5 1 | | | 9/06 |
| | | | | | 3 7 0 N |
| | | | | | 3 5 1 H |
| | | | | | 4 2 0 J |

審査請求 未請求 請求項の数 4 FD (全 34 頁)

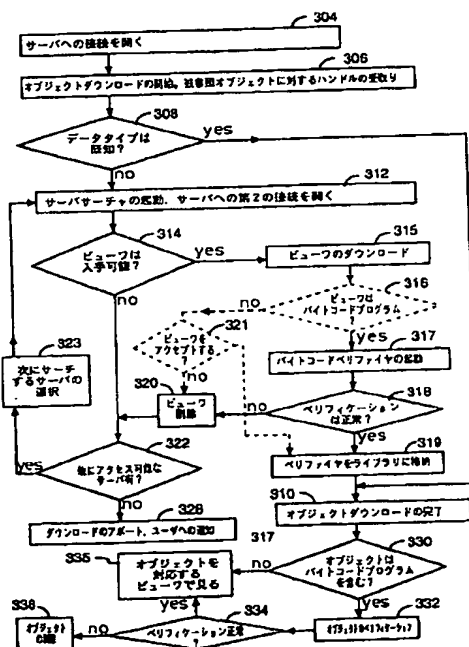
| | | | |
|-------------|------------------|---------|---|
| (21)出願番号 | 特願平7-349164 | (71)出願人 | 595034134 サン・マイクロシステムズ・インコーポレ イテッド Sun Microsystems, Inc. アメリカ合衆国カリフォルニア州94043- 1100・マウンテンビュー・ガルシアアベニ ュー 2550 |
| (22)出願日 | 平成7年(1995)12月20日 | (72)発明者 | ジェイムス・エイ・ゴスリング アメリカ合衆国カリフォルニア州94062・ ウッドサイド・ビーオーボックス 620509 |
| (31)優先権主張番号 | 08/359, 884 | (74)代理人 | 弁理士 大島 陽一 (外1名) |
| (32)優先日 | 1994年12月20日 | | |
| (33)優先権主張国 | 米国(US) | | |

(54)【発明の名称】 分散型コンピュータシステム及びその動作方法

(57) 【要約】

【課題】 参照しようとするオブジェクトのビューを自動的にダウンロードするとともにロードしたプログラムの健全性をベリファイするコンピュータを含む分散型コンピュータシステム及びその動作方法を提供すること。

【解決手段】 本発明は、他のサーバなどに格納されているデータ（オブジェクト）をネットワークを介してロードして参照するとき、そのオブジェクトに対応するビューを自動的にサーチし、他のサーバなどで適切なビューが見つかった場合、そのビューを起動する前に、保全性を確認するためベリフィケーションを行うことを主な特徴とする。本発明は、特にバイトコード言語で書かれたプログラムに対するデータタイプとスタックの使用に関するベリフィケーションに重点を置いている。



【特許請求の範囲】

【請求項 1】 複数の個別のコンピュータを有する分散型コンピュータシステムの動作方法であって、

(a) 第 1 コンピュータに、各々関連するデータタイプのオブジェクトを見るための複数のビューワプログラムを格納する過程と、

(b) 前記第 1 コンピュータに於いて、第 2 コンピュータ内に格納されているオブジェクトへのリファレンスを選択する過程と、

(c) 前記第 1 コンピュータに於いて、前記リファレンスの選択に応じて、前記第 1 コンピュータと前記第 2 コンピュータとの間に第 1 接続リンクを確立し、前記オブジェクトに関連するデータタイプ情報の検索を含む、前記第 2 コンピュータからの前記オブジェクトの検索を開始する過程と、

(d) 前記第 1 コンピュータに於いて、前記第 1 コンピュータに格納された前記ビューワプログラムの中に、前記検索されたデータタイプに対応するビューワプログラムが含まれているかどうか判定する過程と、

(e) 前記過程 (d) での判定結果が否定の場合、前記検索されたデータタイプに対応するビューワプログラムが、前記第 1 コンピュータを含む他のコンピュータのセット内に格納されているかどうか判定する過程と、

(f) 前記過程 (e) での判定結果が肯定の場合、(f 1) 前記第 2 データタイプに対応する前記ビューワプログラムのコピーを前記第 1 コンピュータにロードする過程と、(f 2) 前記コピーされたビューワプログラムに対しベリフィケーションプロシージャを実行して、前記コピーされたビューワプログラムが予め定められたオペランドスタック使用基準に合うかどうか判定する過程と、(f 3) 前記過程 (f 2) での判定結果が肯定の場合、前記第 2 オブジェクトを見ることができるよう前記コピーされたビューワプログラムを実行する過程とを含むことを特徴とする方法。

【請求項 2】 前記過程 (e) が、前記検索されたデータタイプに対応するビューワプログラムが前記第 2 コンピュータに格納されているかどうか最初に判定し、前記判定の結果が否定の場合、前記第 2 データタイプに対応するビューワプログラムが予め定められた他のコンピュータのセットの何れかの中に格納されているかどうか判定し、前記第 2 の判定結果が肯定の場合、前記過程 (f 1) 乃至 (f 3) を実行する過程を含んでいることを特徴とする請求項 1 に記載の方法。

【請求項 3】 複数の個別のコンピュータを有する分散型コンピュータシステムであって、第 1 コンピュータと第 2 コンピュータを含み、前記第 1 コンピュータが、オブジェクト及びビューワプログラム格納用第 1 メモリを有し、格納されている前記オブジェクトの各々は関連するデータタイプ情報を含み、

前記第 2 コンピュータが、ビューワプログラム格納用第 2 メモリと、前記第 1 コンピュータの前記第 1 メモリ内に格納されている前記オブジェクトの一つへのリファレンスを、前記第 2 コンピュータのユーザが選択することができるようにするためのユーザインタフェース制御プログラムと、前記オブジェクトリファレンスの選択に応じて、前記第 2 コンピュータと前記第 1 コンピュータとの間に第 1 接続リンクを確立し、前記オブジェクトに関連するデータタイプ情報の検索を含む、前記第 1 コンピュータからの前記オブジェクトの検索を開始するコンピュータ間リンク制御プログラムとを含み、前記第 1 及び第 2 コンピュータに含まれる各ビューワプログラムは、対応するデータタイプのオブジェクトを見ることができるとし、

前記ユーザインタフェース制御プログラムが、前記第 2 コンピュータに含まれている前記ビューワプログラムに、前記検索されたデータタイプに対応するビューワプログラムが含まれているかどうか判定して、判定結果が否定の場合、前記検索されたデータタイプに対応するビューワプログラムを前記第 1 コンピュータを含む他のコンピュータのセットの中から見つけようと試みるビューワサーチ命令を含み、

前記コンピュータ間リンク制御プログラムが、前記ビューワサーチ命令が前記検索されたデータタイプに対応するビューワプログラムを前記他のコンピュータ内に見つけたとき、そのビューワプログラムのコピーを前記第 2 コンピュータ内にロードするためのビューワダウンロード命令を含んでいることを特徴とするシステム。

【請求項 4】 前記第 2 コンピュータが、更に、前記コピーされたビューワプログラムが、予め定められたオペランドスタック使用基準に合うかどうか判定するためのベリフィケーションプロシージャと、前記ベリフィケーションプロシージャが前記コピーされたビューワプログラムが前記予め定められた基準に合うと判定したとき、前記第 2 オブジェクトを見るための前記コピーされたビューワプログラムの実行をイネーブルするプログラムイネーブル命令とを含むことを特徴とする請求項 3 に記載のシステム。

【発明の詳細な説明】

【0001】

【発明が属する技術分野】本発明は、一般に、個別の基本マシン命令セットを用いる複数のコンピュータプラットフォーム上のコンピュータソフトウェアの利用法に関する。特に、ネットワークサーバまたは他のソースから得られるコンピュータソフトウェアの保全性をベリファイ（検証）する方法に関する。

【0002】

【従来の技術】図 1 に一般的に示されているように、典型的な従来のネットワーク化されたコンピュータシステム

ム 100 に於いて、第 1 コンピュータ 102 は、第 2 コンピュータ 104 にあるコンピュータプログラム 103 をダウンロードすることができる。この例では、第 1 ユーザノード 102 は、典型的には、中央演算装置 (CPU) 106、ユーザインタフェース 108、プログラム実行用の主メモリ 110 (例えば RAM)、オペレーティングシステム 113、文献及び他のデータを格納するための補助記憶装置 112 (例えばハードディスク)、及びコンピュータネットワーク 120 (例えばインターネット、ローカルエリアネットワークまたはワイドエリアネットワーク) に接続するためのモデムなどのコミュニケーションインタフェース 114 を有するユーザワークステーションである。コンピュータ 102 及び 104 は、“ネットワーク上のノード” または “ネットワークノード” と呼ばれることもある。

【0003】第 2 コンピュータ 104 はネットワークサーバであるのが一般的だが、第 2 ユーザワークステーションであっても良く、通常、第 1 コンピュータと基本的に同じコンピュータ要素を含んでいる。

【0004】従来、第 1 コンピュータ 102 が第 2 コンピュータ 104 からコンピュータプログラム 103 のコピーをダウンロードした後に、第 1 コンピュータ 102 のユーザが、ダウンロードされたプログラム 103 の保全性をベリファイするのを支援するための標準的なツールは実質的にない。特に、第 1 コンピュータのユーザが、ダウンロードされたプログラムのソースコードを調べない限り、従来技術のツールを用いて、ダウンロードされたプログラム 103 がスタックに対しアンダーフローやオーバフローとならないか、或いはダウンロードされたプログラム 103 がユーザコンピュータのファイル

や他のリソースに悪影響を与えないかどうかを判断することは実質的に不可能である。

【0005】あるコンピュータから別のコンピュータへのコンピュータソフトウェアのダウンロードに関する別の問題は、個別の基本マシン命令セットを用いているコンピュータプラットフォーム間のコンピュータソフトウェアの転送の問題である。従来も、プラットフォームに依存しないコンピュータプログラムやプラットフォームに依存しないコンピュータプログラミング言語の例はいくつかある。従来技術に欠けているのは、ネットワークサーバまたは他のソースから得られる、プラットフォームに依存しないコンピュータソフトウェアの保全性を、そのソフトウェアを受け取った人がベリファイするのを可能にする信頼の置ける自動ソフトウェアベリフィケーション (verification) ツールである。

【0006】本発明の別の側面は、遠隔からダウンロードするオブジェクトまたはファイルをユーザが選択した後に、そのオブジェクトまたはファイルに関連するソフトウェアを自動的にダウンロードするための方法に関する。例えば、“ワールドワイドウェブ (World Wide We

b: WWW)” として知られている、インターネットで広く用いられている機能があるが、この WWW 上で、ある文献を検索しているとき、その文献のあるページに、他の文献またはオブジェクトへのリファレンス (参照) が含まれていることがある。ユーザは、所与のオブジェクトを選択することによって、関連するハイパーリンクを介し、それらの文献またはオブジェクトにアクセスすることができる。そのような選択は、通常、ユーザがワークステーションノードのグラフィカルユーザインタフェースを使い、ポインタデバイスを用いてハイパーリンクを表すグラフィックイメージを指定し、ポインタデバイス上のボタンを押すことによって行われる。ハイパーリンクの選択に応答して、ユーザのウェブアクセスプログラムは、参照したい文献またはオブジェクトが格納されているサーバ (その時見ている文献またはオブジェクト内のハイパーリンクに組み込まれているデータによって示されている) への接続を開き、それらの文献またはオブジェクトをダウンロードする。しかしながら、ダウンロードされる文献またはオブジェクトのデータタイプが、ユーザのウェブアクセスプログラムにとって既知でない場合、ユーザはダウンロードした文献を見ることができないか、或いは見ることはできても用いることができない。

【0007】このようなことが起きると、ユーザは、しばしば文献またはオブジェクトを検索してきたサーバや、或いは他のサーバのプログラムライブラリを調べて、ダウンロードした文献またはオブジェクトに対応するビューワ (viewer) をマニュアルで見つけようと試みるかもしれない。ユーザのコンピュータプラットフォームとコンパチビリティ (compatibility) のあるビューワが見つかり、ユーザはそのビューワをダウンロードし、それを起動 (実行) して、前にダウンロードしたオブジェクトを見ようとするだろう。しかしながら、出所のはっきりしないビューワを実行することには、かなり大きなリスクがある。例えば、ダウンロードしたビューワプログラムが “ウィルス” に感染しており、ユーザのコンピュータの保全性を損なうかもしれない。或いは、ユーザの期待に反して、ダウンロードしたプログラム自身がリソースへのアクセス及び/またはユーザコンピュータのデータの破壊を行う可能性もある。本発明は、文献乃至オブジェクトに対応するビューワを自動的にダウンロードするとともに、ダウンロードしたビューワを起動する前に、そのプログラムの保全性を自動的にベリファイすることによってこのような問題を解消するものである。

【0008】

【発明が解決しようとする課題】上述したように、本発明の主な目的は、参照しようとする、他のコンピュータ (サーバなど) に含まれる文献またはオブジェクトに対応するビューワを自動的にダウンロードするとともに、

ダウンロードしたビューワを起動する前に、そのプログラムの保安全性を自動的にペリファイするコンピュータを含む分散型コンピュータシステム及びその動作方法を提供することである。

【0009】

【課題を解決するための手段】本発明は、遠隔のコンピュータノードからオブジェクト及びオブジェクトビューワを検索（即ちダウンロード）したり、オブジェクトを見るべく、格納したオブジェクトビューワを起動したりするための“クラスローダ（class loader）”に関する。ユーザが、例えばWWWのハイパーリンク機能を用いることによって見たいオブジェクトを選択すると、従来と同様に、参照されるオブジェクト（被参照オブジェクトとよぶこともある）のダウンロードが開始される。しかしながら、本発明のクラスローダは、オブジェクトのダウンロードプロセス開始時に受け取るデータタイプ情報を用いて、参照するオブジェクトに対応するビューワが、ユーザワークステーションに於いて入手可能かどうかをまず調べる。

【0010】適切なビューワがそのワークステーションに於いて得られない場合、クラスローダは、オブジェクトをダウンロードしているサーバ、或いはユーザのワークステーションが認識している他の適切なサーバから適切なビューワを自動的に見つける。クラスローダは、見つけたビューワをダウンロードした後、このビューワを起動する前に、その保安全性をペリファイするためプログラムペリフィケーションプロシージャを呼び出す。いったんペリファイされると、ビューワはユーザのローカルビューワライブラリに加えられ、被参照オブジェクトのダウンロードは終了し、ダウンロードしたオブジェクトを見るためのビューワの起動がイネーブルされる。

【0011】適切なビューワが見つからないか、或いは見つけた唯一のビューワがペリフィケーションプロシージャを通過できない場合は、参照しようとしていたオブジェクトのダウンロードはアボートされる。

【0012】本発明では、限定された、データタイプに依存したバイトコードセットを用いたバイトコード言語（“OAK言語（OAK language）”として商品化されている）で書かれたコンピュータプログラムの保安全性をペリファイする。この言語で使用可能なソースコードのバイトコードは、（A）処理可能なデータタイプについて制約があるスタックデータ消費型バイトコードである、（B）スタックデータは使用しないが、データタイプが既知であるデータをスタックに加えたり、データタイプに関係なくスタックからデータを取り出したりすることによってスタックに影響を与える、或いは（C）スタックデータの使用も、スタックへのデータの付加もしないのいずれかである。

【0013】本発明によると、バイトコードプログラムを実行する前に、データタイプがマッチしていないデー

タを処理しようとする命令がないかどうか、あるいはその指定されたプログラム内のバイトコード命令の実行によってオペランドスタックのアンダーフローやオーバーフローが生じないかどうかをチェックし、そのようなプログラムを使用するのを避けるための方法及びペリファイヤが提供される。

【0014】本発明のバイトコードプログラムペリファイヤは、指定されたバイトコードプログラムを実行するとき、プログラムオペランドスタック内に格納されるデータを示すスタック情報を一時的に格納するための仮想スタック（virtual stack）を含んでいる。このペリファイヤは、プログラムの各ポイントに於いてオペランドスタックに格納されるであろうデータの数、順番、及びデータタイプを示すように仮想オペランドスタックを更新しながらプログラムの各バイトコード命令を順次処理していくことによって、指定されたプログラムを処理する。このペリファイヤは、仮想スタックの情報と、各バイトコード命令のデータタイプに関する制約とを比べて、プログラムを実行したときオペランドスタックにバイトコード命令のデータタイプに関する制約に適合しないデータが含まれるようなことがないかどうか判定するとともに、指定されたプログラム内のバイトコード命令によってオペランドスタックのアンダーフローやオーバーフローが生じるようなことはないかどうかチェックする。

【0015】バイトコードプログラムの命令の流れを細かく分析しなくていいように、またバイトコード命令を何度もペリファイしなくていいように、指定されたプログラム内に於いて別個の2以上のバイトコードの実行の直後に実行され得る全てのポイント（多重エントリポイントと呼ぶ）がリストアップされる。一般に、そのような先行する2以上の別個のバイトコードの少なくとも一つは、ジャンプ／分岐バイトコードである。指定されたプログラムの処理時、ペリファイヤは、各多重エントリポイントの直前（即ち、先行するバイトコード命令の一つの後）の仮想オペランドスタックの“スナップショット”をとり、このスナップショットを、同じ多重エントリポイントに対する他の先行するバイトコード命令の各々を処理した後の仮想スタックステータスと比較して、仮想スタックステータスが同一でない場合はプログラムフォールト（program fault）を生成する。

【0016】

【発明の実施の形態】本発明の好適実施例について詳細に説明する。これらの例は添付の図面に図示されている。本発明を好適実施例に基づいて説明するが、理解されるように、それらは本発明をそれらの実施例に限定することを意図するものではない。本発明は、特許請求の範囲によって画定される本発明の精神及び範囲に含まれるべき変形変更及び等価物を含むものである。

【0017】図2に示す分散型コンピュータシステム2

00を参照されたい。第1コンピュータノード202は、例えばインターネットのようなコンピュータコミュニケーションネットワーク216を介して第2コンピュータノード204とつながっている。第1コンピュータノード202は、中央演算装置(CPU)206、ユーザインタフェース208、主メモリ(RAM)210、補助記憶装置(ディスク記憶装置)212、及び第1コンピュータノード202をコンピュータコミュニケーションネットワーク216につなげるためのモデムその他のコミュニケーションインタフェース214を含んでいる。ディスク記憶装置212は、データファイルその他の情報と共に、CPU206によって実行されるプログラムを格納する。

【0018】第2コンピュータノード204(ここでは、ファイルや他の情報のサーバとして構成されているものとする)は、CPU218、ユーザインタフェース220、主メモリ(RAM)222、補助記憶装置(ディスク記憶装置)224、及びコンピュータコミュニケーションネットワーク216と第2コンピュータノードとを接続するためのモデムその他のコミュニケーションインタフェース226を含んでいる。ディスク記憶装置224は、ファイル及び/またはオブジェクトディレクトリ228(ディスクディレクトリまたはカタログと呼ばれることもある)を含んでいる。ディスクディレクトリ228は、補助記憶装置224内に格納されている情報、即ち、CPU218によって実行されたり他のコンピュータノードに配布されたりするプログラム234、ビューワライブラリ232、及びオブジェクト230の位置を示す。

【0019】第1コンピュータノード202及び第2コンピュータノード204は、それぞれが異なるコンピュータプラットフォーム及びオペレーティングシステム236、237を用い、これらの2つのコンピュータノードの一方で実行されるオブジェクトコードプログラムが、他方のコンピュータでは実行できないことがあってもよい。例えば、サーバノード204をUnixオペレーティングシステムを用いたサン・マイクロシステムズ社のコンピュータとし、ユーザワークステーションノード202を80486マイクロプロセッサとマイクロソフト社のDOSオペレーティングシステムを用いたIBM互換機とすることもできる。更に、同じネットワークに接続され、同じサーバ204を利用する他のユーザワークステーションに於いて、様々なコンピュータプラットフォーム及びオペレーティングシステムを用いてもよい。

【0020】これまで、様々なタイプのコンピュータを有するネットワークでソフトウェアを配布するためのサーバ204は、各コンピュータプラットフォーム(例えば、Unix、ウィンドウズ、DOS、マッキントッシュなど)に対して個別のソフトウェアライブラリ(例え

ば個別のビューワライブラリ232)を有していた。その結果、様々なシステムのユーザの要望に応えるため、サーバは同じコンピュータプログラムの様々なバージョン(例えば238、239)と、複数のオブジェクトビューワ(例えば241、243)の両方を、各コンピュータプラットフォームタイプに対し一つずつ格納する必要があった。しかしながら、本発明を用いると、種類のバイトコードバージョンのプログラムを配布することによって、多様なユーザをサポートすることができる。

【0021】図3に、本発明に従った分散型コンピュータシステム250を示す。第1コンピュータノード252は、例えばインターネットのようなコンピュータコミュニケーションネットワーク266を介して第2コンピュータノード254につながっている。ここでもまた、従来と同様に、第1コンピュータノード252と第2コンピュータノード254が、別個のコンピュータプラットフォーム及びオペレーティングシステム255、256を用い、一方のコンピュータで実行されるオブジェクトコードプログラムが他方のコンピュータで実行できなくてもよい。例えば、図2を参照して上述したように、サーバノード254がUnixオペレーティングシステムを用いたサンマイクロシステムズ社のコンピュータで、ユーザワークステーションノード252が80486マイクロプロセッサとマイクロソフト社のDOSオペレーティングシステムを用いたIBM互換機であってもよい。第1コンピュータノード252は、CPU257、ユーザインタフェース258、主メモリ(RAM)260、補助記憶装置(ディスク記憶装置)262、及びコンピュータコミュニケーションネットワーク266に第1コンピュータノード252を接続するためのモデムその他のコミュニケーションインタフェース264を含んでいる。ディスク記憶装置262は、CPU257によって実行されるプログラムを格納する。これらのプログラムの少なくとも一つは実行可能形式のバイトコードプログラム267である。説明のため、第1コンピュータノード252が、第2コンピュータノード254からコンピュータコミュニケーションネットワーク266を介してバイトコードプログラム267を受け取ることとする。その詳細について、クラスローダと共に以下に説明する。

【0022】好適実施例では、バイトコードプログラムはOAKアプリケーションとして書かれており、コンパイルまたはインタープリットされて、一連の実行可能な命令になる。OAK命令セットの全ソースコードバイトコード命令(source code bytecode instructions)のリストを表1に示す。OAK命令セットは、データタイプに依存する(即ちデータタイプが特定されている)バイトコード命令によって特徴付けられる。詳述すると、OAK命令セットでは、異なるデータタイプに対する同じ基本操作に別個の操作コード(opcode)を付すことに

よって、それらを区別している。その結果、同じ基本機能（例えば2つの数字の加算）を実行するために、それぞれ対応する特定のデータタイプのデータだけを処理する複数のバイトコードが命令セット内に含まれている。更に、OAK命令セットに含まれていない命令についても注意されたい。例えば、OAK言語命令セットには“計算されたgoto (computed goto)”命令はなく、オブジェクトリファレンスを変更したり、新たなオブジェクトリファレンスを生成する命令もない（既存のオブジェクトリファレンスのコピー命令は除く）。OAK命令セットに関するこれらの2つの制約は、他の制約とともに、OAK命令セットに含まれるデータタイプが特定された命令と同じようにデータを用いるバイトコードプログラムであれば、ユーザのコンピュータシステムの健全性を損なうことはないということを保証するのに役立っている。

【0023】好適実施例では、使用可能なデータタイプは、整数、倍長整数 (long integer)、ショートインテジャ (16ビット信号の整数)、単精度浮動小数点数、倍精度浮動小数点数、バイト、キャラクタ、及びオブジェクトポインタ（本明細書中では、オブジェクトリファレンスと呼ぶこともある）がある。各“オブジェクトリファレンス”データタイプはデータタイプの一部としてオブジェクトクラス仕様 (object class specification) を含むことができるため、“オブジェクトリファレンス”データタイプに含まれるデータのサブタイプの数には実質的に制限がない。更に、プログラム内で用いられる定数もデータタイプを有しており、好適実施例で使用される定数のデータタイプには、上述したデータタイプに加えて、class、fieldref、methodref、string、及びAscizが含まれる。それらは全て特定の目的を持った2バイト以上の定数データを表す。

【0024】データタイプに依存しないバイトコードの中には、スタック操作機能を有するものがいくつかある。スタック操作機能には(A)スタック上の1または複数のワードを複製して、それらをスタック内の特定の位置に配置し、それによってデータタイプが既知のより多くのスタックアイテム (stack item) を生成する機能、(B)1または複数のアイテムをスタックから消去する機能などが含まれる。他の少数のデータタイプに依存しないバイトコードには、スタック上のワードを全く用いずスタックを変化させなかったり、あるいは、スタック上にあるワードを使用することはないがスタックにワードを加えたりするものがある。これらのバイトコードは、それらが実効される前のスタックの内容についてデータタイプに関する制約はないが、スタック内のアイテムのデータタイプに関してスタックの内容を変化させる。ただし、その変化のさせかたは全く予測可能である。従って、スタック内の全てのオペランドのデータタイプ及びその数は、100%の精度で常に予測（即ち計

算）可能である。

【0025】第2コンピュータノード254（ここではファイルまたは他の情報サーバとして構成されているものとする）は、CPU268、ユーザインタフェース270、主メモリ (RAM) 272、補助記憶装置（ディスク記憶装置）274、及びコンピュータコミュニケーションネットワーク266に第2コンピュータノードを接続するためのモデムその他のコミュニケーションインタフェース276を含んでいる。ディスク記憶装置274は、ディスクディレクトリ280、第1オブジェクト283を含むオブジェクト282、ビューライブラリ284、及びCPU268によって実行されたり、及び/または他のコンピュータノードに配布されたりするプログラム286（それらのうちの少なくとも一つはコンピュータノード252に転送されるバイトコードプログラム267である）を含んでいる。

【0026】図3に示されているように、第1コンピュータノード252の補助記憶装置262内には、他のコンピュータノードからオブジェクトビューワやオブジェクトを検索（即ちダウンロード）し、格納したオブジェクトビューワを呼び出してオブジェクトを見るためのクラスローダプログラム296が格納されている。また、クラスローダ296は、ダウンロードされたオブジェクトビューワがユーザによって起動される前に、そのビューワを（エンドユーザワークステーションノードに於いて）自動的にベリファイし、その健全性をチェックする。

【0027】この特許出願の目的に於いては、対応するビューワを用いることによって“見る”ことが可能な“オブジェクト”は、(A)JPEG、GIF、MPEG、あるいはMPEG2データといった特定のタイプまたはフォーマットのデータを含むファイルその他のデータ構造の、メソッド (method) またはソフトウェアが組み込まれていないデータオンリー型オブジェクト (data-only type of object)、あるいは(B)1または複数のメソッドが組み込まれており、オプションとしてデータも含み得るファイルその他のデータ構造の、メソッド格納型オブジェクト (method-storing object) のいずれかである。例えばJPEGやGIFのようなイメージデータタイプを格納しているデータオンリー型オブジェクトを見るため、あるいは、例えばMPEGやMPEG2のようなビデオプログラムデータタイプを格納しているデータオンリー型オブジェクトを見るためには、特定のビューワが必要であろう。他の例として、データの図表を見るためのビューワや、（解読のキーがユーザに既知の場合）暗号化されたデータを見るためのデータ解読ソフトウェアが含まれたビューワなどがあり得る。

【0028】更に、様々なタイプの内部プログラムを用いたメソッド格納型オブジェクトに対して特定のビューワが必要となることがある。例えば、様々なメソッド格

納型オブジェクトの様々なタイプの内部プログラムが、特定の記述言語を用いていたり、ユーティリティプログラムの様々なライブラリの使用を仮定していたりすることによって、特定のビューが必要となることがある。

【0029】“ビュー（インタープリタと呼ぶこともある）”は、指定されたオブジェクト内のデータ及び／または命令のデコードを行う。また、一般に特定のデータタイプまたはクラスのオブジェクトを使用可能にするために必要とされている計算及び処理を全て実行する。本発明では、そのようなオブジェクトビューは、ソースコードバイトコード言語（source code bytecode language）で書かれたバイトコードプログラムであり、各オブジェクトビューの保天性は、バイトコードプログラムベリファイヤ240を実行することによって、エンドユーザにより個別にベリファイすることができる。バイトコードプログラムのベリフィケーションについて、以下により詳細に説明する。

【0030】分散型コンピュータシステム250は、本発明によるプラットフォームに依存しないオブジェクトビューを含むことができるとともに、プラットフォームに依存し、本発明によるバイトコードプログラムベリファイヤ240及びクラスローダ296を用いてベリファイすることのできない他のオブジェクトビューも含むことができるという点に注意されたい。このようなハイブリッドシステムでは、本発明による自動ビュー保天性ベリフィケーションは、バイトコードビュープログラムに対しては利点があるが、他のビュープログラムに対しては利点がない。

【0031】クラスローダ296は、遠隔にあるサーバからのオブジェクトビュー及びオブジェクトのロード及びベリフィケーション用実行可能プログラムである。例えば、インターネットのWWW上である文献を検索するとき、その文献のあるページに他の文献またはオブジェクトへのリファレンスが含まれていることがある。ユーザは、そのような他の文献またはオブジェクトに、関連するハイパーリンクを介して所与のオブジェクトを選択することによってアクセスすることができる。そのような選択操作は、通常、ワークステーションノード上のグラフィカルユーザインタフェースを利用して、ポインタデバイスを用いてハイパーリンク選択を表すグラフィックイメージを指し示し、ポインタデバイス上のボタンを押すことによって、ユーザにより為される。

【0032】選択プロセスに於いて、そのとき見ている文献またはオブジェクトにリファレンスが含まれている他の文献またはオブジェクトの中には、データタイプがユーザのワークステーションに既知でないものもあるだろう。本発明のクラスローダは“既知でない”データタイプに対応するビューを見つけると共に、ダウンロードされたバイトコードプログラムがユーザによって実行される前に、それらの保天性をベリファイするのに用い

られる。

【0033】クラスローダ296には3つの主要な機能がある。第1に、クラスローダはダウンロードされたオブジェクト（及びそれらの関連するバイトコードプログラム）のデータタイプをチェックし、ユーザワークステーションが、その固有の記憶装置262に格納されている“ビューライブラリ”298内に、対応するビューを有しているかどうかを調べる。第2に、適切なビューを見つけることができなかった場合、クラスローダは、ソースサーバと、認識している他のサーバとに対してサーチルーチンを実行し、適切なビューを見つけてダウンロードすることを試みる。ビューを見つけることができなかった場合は、ダウンロードされているオブジェクト及び／またはバイトコードプログラムを、適切なビューがないという理由でリジェクトする。最後に、リモートソースに適切なビューが見つかった場合、クラスローダは、バイトコードベリファイヤ240を起動し、バイトコードプログラムインタープリタ242によってビューが実行される前に、またはバイトコードプログラムコンパイラ244によってコンパイルされる前に、ダウンロードされたビューをチェックする。ベリフィケーション後、ダウンロードされたビューは、ユーザのローカルビューライブラリ298内に格納される。

【0034】図3、図4及び付録1には、クラスローダプログラム296の動きが、関連するオブジェクトを介したバイトコードプログラムの検索に対し詳細に記載されている。付録1は、クラスローダプログラムを表す疑似コード（pseudocode）のリストである。付録1で用いられている疑似コードは、本質的には、従来の汎用コンピュータ言語を用いたコンピュータ言語と同等である。本願中で用いられている疑似コードは、本発明を説明する目的で作られたものであり、本分野で標準的な知識を有するコンピュータプログラマであれば容易に理解することができるように意図して書き下してある。

【0035】図4に示されているように、ユーザワークステーション252は、ダウンロードされるオブジェクト283が格納されているサーバ254への接続を開き、ダウンロードプロセスを開始する（304）。クラスローダ296は、オブジェクトのハイパーリンク選択により、オブジェクトバイトコードプログラムの転送を開始し、サーバ254は参照されるオブジェクトに対する“ハンドル（handle）”を、ユーザワークステーション252に転送する（306）。ハンドルは、参照されるオブジェクト本体に先だって検索される。また、ハンドルには、オブジェクトのデータタイプ（オブジェクトクラスと呼ぶこともある）を含む、被参照オブジェクトの属性に関する情報が含まれる。

【0036】最初のチェックに於いて、検索されるオブジェクトに関連するデータタイプが、ユーザシステムに

既知であるかどうかが判定される(308)。詳述すると、クラスローダは、ユーザワークステーション252の補助記憶装置262内にあるビューワライブラリ298をサーチして、特定のデータタイプを有するこのオブジェクトに対する適切なビューワが、アクセス可能かどうかチェックする。ビューワライブラリ298には、その時々においてユーザワークステーションによってアクセス可能な全てのデータタイプのビューワと、それらのメモリ内の位置とを示すリストが含まれている。このようにして、クラスローダは、参照されるオブジェクト本体を実際にダウンロードする前に、最初のハンドシェイクプロセスにおいてダウンロードされるオブジェクトを前処理し、そのオブジェクトとユーザワークステーションプラットフォームのコンパチビリティをチェックする。適切なビューワが見つかった場合、クラスローダは被参照オブジェクトのダウンロードを完了する(310)。

【0037】適切なビューワが、ビューワライブラリ298内に見つからなかった場合(即ち、選択されたオブジェクトのデータタイプがユーザワークステーション252に既知でないことを意味する)、クラスローダは適切なビューワのサーチを行う。通常、ビューワのサーチを試みる最初の場所は、選択されたオブジェクトが格納されているのと同じサーバである。従って、クラスローダは、被参照オブジェクトが格納されているのと同じサーバに対して第2の接続を開き(312)、指示されているデータタイプに対応するビューワをリクエストする(314)。このサーバが適切なビューワを含んでいる場合、そのビューワはユーザワークステーションにダウンロードされる(315)。

【0038】ビューワのダウンロードが完了すると、ダウンロードされたビューワがバイトコードプログラムの場合(316)、クラスローダは、バイトコードプログラムベリファイヤ240を呼び出して、そのビューワプログラムのベリフィケーションを開始する(317)。バイトコードプログラムベリファイヤ240は実行可能プログラムであり、CPU257によってバイトコードプログラムが実行される前に、指定されたバイトコード(ソース)プログラムを、スタック操作の適正さや、オペランドデータタイプコンパチビリティについてベリファイする。バイトコードベリファイヤプログラム240の動作については、後により詳細に説明する。ベリフィケーション結果が正常の場合(318)、サーバサーチ(server searcher)はベリファイされたオブジェクトビューワをビューワライブラリ298内に格納し、新たなデータタイプビューワが使用可能になったことが反映されるようにライブラリ内のディレクトリを更新する(319)。ベリフィケーションが正常でなかった場合、ダウンロードされたビューワは削除される(320)。

【0039】本発明の実施例には、自動ダウンロードが可能で、かつベリファイ可能なオブジェクトビューワとベリファイ不能なオブジェクトビューワの両方を使用することができるものもある。このような実施例では、オブジェクトビューワのダウンロード(315)の後、ダウンロードされたオブジェクトビューワがバイトコードプログラムでない場合(316)、そのオブジェクトビューワをアクセプトするか否かに対し判定がなされる(321)。例えば、そのオブジェクトビューワをアクセプトするかどうかユーザに質問してもよく、あるいはそのようなオブジェクトビューワをアクセプトするかどうかを予め決めてコンフィギュレーションファイル(configuration file)に含ませておいてもよい。ベリファイ不能なオブジェクトビューワがアクセプトされる場合、そのオブジェクトビューワはビューワライブラリ内に格納される(319)。アクセプトされない場合、ダウンロードされたビューワは削除される(320)。

【0040】サーバもユーザワークステーションも適切なビューワを格納しておらず、ステップ308及び314で、選択されたオブジェクトに対し用いるのに適切なビューワが見つからなかった場合、クラスローダは、ユーザワークステーションに認識されている他のサーバまたはリモートユーザワークステーション(例えば認識されているサーバのリスト327)を含むように探索範囲を広げる。再度図3を参照されたい。図3に示されている第2サーバ324は、ビューワライブラリ326を有する補助記憶装置(ディスク記憶装置)325を含んでいる。適切なビューワが第2サーバ324のビューワライブラリ326内で見つかった場合、クラスローダは、上述したステップ315〜321に従って、そのビューワプログラムをダウンロードしベリファイする。クラスローダはこのプロセスを繰り返し、全ての認識されているリソースについて調べ尽くすか、または適切なビューワを見つけてベリファイするまで、順次サーバをチェックする。最終的に、適切なビューワを見つけることができなかった場合、被参照オブジェクトのダウンロードはアボートされ、被参照オブジェクトに対応するビューワが見つからなかったことをユーザに通知するべく、ユーザメッセージが生成される(328)。

【0041】上述したように、適切なオブジェクトビューワが予めユーザワークステーション上のビューワライブラリ298に格納されていた場合(308)、或いは適切なオブジェクトビューワのダウンロード、ベリフィケーション、及びユーザビューワライブラリへの組み込みが成功した場合、選択されたオブジェクトのロードが完了する(310)。ダウンロードされたオブジェクトに1または複数のバイトコードプログラムが組み込まれている場合、従ってメソッド格納型オブジェクトである場合(330)、ダウンロードされたオブジェクト内のバイトコードプログラムは、これらの組み込みプログラ

ムに関し、バイトコードベリファイヤを実行することによりベリファイされる(332)。組み込みプログラムの処理が終了し、ベリファイヤが“成功”リターンコードを生成する場合(334)、ダウンロードされたオブジェクトは対応するオブジェクトビューワによって見ることができる(335)。ベリファイヤの要求に従わないプログラムを検出することによって、ベリファイヤが組み込みプログラムの処理をアボートした場合(334)、ダウンロードされたオブジェクトは削除され(336)、適切なユーザメッセージが生成される。

【0042】ダウンロードされたオブジェクトに組み込みバイトコードプログラムが含まれていない場合(330)、ステップ332乃至334はスキップされ、オブジェクトは適切なビューワによって見ることができる(335)。

【0043】再度図3を参照されたい。図示されているように、第1コンピュータノード252は、その補助記憶装置262内に、指定されたバイトコードプログラムの保性をベリファイするためのバイトコードベリファイヤプログラム240と、指定されたバイトコードプログラムを実行するためのバイトコードインタプリタ242を格納している。別の手段として、または追加して、第1コンピュータノード252は、ベリファイされたバイトコードプログラムをオブジェクトコードプログラムに変換し、インタプリタ242より効率的にバイトコードプログラムを実効するバイトコードコンパイラ244を格納している。

【0044】バイトコードベリファイヤ240は実行可能プログラムであり、バイトコードプログラムがバイトコードインタプリタ242の制御下でCPU257によって実行される前に(またはバイトコードプログラムがコンパイラ244によってコンパイルされる前に)、指定されたバイトコード(ソース)プログラムに於けるスタック操作の適正さや、オペランドデータタイプコンパチビリティをベリファイする。各バイトコードプログラム267(ダウンロードされたオブジェクトベリファイヤを含む)は、対応するベリフィケーションステータス302を有しており、この値はプログラムが他の場所からダウンロードされたとき、最初“偽(False)”にセットされる。プログラムに対するベリフィケーションステータス302は、プログラムがベリファイされ、ベリファイヤ240によって行われるデータタイプテスト及びスタックの使用に関するテストの全てに適合したときのみバイトコードベリファイヤ240によって“真(True)”にセットされる。

【0045】バイトコードプログラムベリファイヤ図5を参照されたい。バイトコードプログラムベリファイヤ240の処理について、特定のバイトコードプログラム340に関連して説明する。ベリファイヤ240は、ベリフィケーションプロセスに於いて必要とする情

報を格納するため、いくつかの一時的なデータ構造(temporary data structure)を使用する。特に、ベリファイヤ240は、スタックカウンタ342、仮想スタック344、仮想ローカル変数アレイ345、及びスタックスナップショット格納構造346を使用する。

【0046】スタックカウンタ342は、仮想スタック操作を追跡し、その時々仮想スタック344のエントリ数が反映されるように、ベリファイヤ240によって更新される。

【0047】仮想スタック344は、実際の実行時にバイトコードプログラム340によってオペランドスタック内に格納される各データに関するデータタイプ情報を格納する。好適実施例では、仮想スタック344は正規のスタックと同様に用いられるが、実際のデータ及び定数を記憶するのではなく、プログラムを実際に実行したときオペランドスタック内に格納される各データに対応するデータタイプを示す値を格納する点が異なる。従って、例えば、もし実際に実行したとき、スタックに3つの値：

HandleToObjctA
5

1

が格納されるとすると、対応する仮想スタックエントリは、

R

I

I

となる。ここで、仮想スタック内の“R”は、オブジェクトリファレンスを示し、仮想スタック内の各“1”

は、整数を示す。更に、この例では、スタックカウンタ342は、仮想スタック344に格納されている3つの値に対応して、値3を格納する。

【0048】各データタイプのデータには、対応する仮想スタックマーカ値(例えば、整数(I)、倍長整数(L)、単精度浮動小数点数(F)、倍精度浮動小数点数(D)、バイト(B)、ショート(S)、オブジェクトリファレンス(R))が割り当てられる。オブジェクトリファレンスに対するマーカ値には、しばしばオブジェクトクラス値が含まれる(例えば、R:point、ここで“point”はオブジェクトクラスである)。

【0049】仮想ローカル変数アレイ345は、仮想スタック344と同じ基本機能を果たす。即ち、仮想ローカル変数アレイ345は、指定されたバイトコードプログラムが使用するローカル変数に対するデータタイプ情報を格納するのに用いられる。データは、しばしばプログラムによってローカル変数とオペランドスタックとの間で転送されるため、そのようなデータ転送を実行するか、あるいはそのようなデータ転送はしないがローカル変数を使用するようなバイトコード命令についてチェックすることにより、各バイトコード命令によってアクセ

スされるローカル変数が、これらのバイトコード命令に関するデータタイプの制約に適合するということを保証することができる。

【0050】動作時、ペリファイヤ240は、スタックからのデータのポップ(pop)をリクエストする各データバイト命令を処理するとともに、同数のデータタイプ値を仮想スタック344からポップする。続いて、ペリファイヤは仮想スタック344から“ポップされた”データタイプ値と、バイトコード命令のデータタイプに関する要求とを比較する。同様に、スタックにデータをプッシュするようにリクエストする各バイトコード命令に対し、ペリファイヤは、対応するデータタイプ値を仮想スタックにプッシュする。

【0051】本発明によるプログラムペリフィケーションの一側面によると、オペランドスタックステータス(operand stack status)に於けるオペランドの数とデータタイプが、ある特定の命令が実行されるとき常に同一となるかという点についてペリフィケーションがなされる。ある特定のバイトコード命令が2以上の異なる命令の直後に処理され得る場合、これらの異なる先行命令の各々を処理した直後の仮想スタックステータスを比較する必要がある。通常、これらの異なる先行命令の内の少なくとも一つは、条件付きまたは無条件ジャンプまたは分岐命令である。上述したような“スタックの一致(stack consistency)”を要求する結果として、各プログラムループ(program loop)によって、オペランドスタック内に格納されるオペランド数の正味の増加や減少が起こるようなことはなくなる。

【0052】スタックスナップショット格納構造346は、スタックカウンタ342及び仮想スタック344の“スナップショット”を格納するのに用いられ、プログラムの様々な点に於ける仮想スタックステータスを効果的に比較することを可能としている。格納されるスタックスナップショットの各々は、例えば：SC、DT1、DT2、DT3、...、DTnのような形であり、ここでSCはスタックカウンタの値、DT1は仮想オペランドスタック内の最初のデータタイプ値、DT2は仮想オペランドスタック内の2番目のデータタイプ値であり、DTnまで同様である。DTnは仮想オペランドスタック内の、全アイテム中最後のアイテムに対するデータタイプ値である。

【0053】スタックスナップショット格納構造346は、ディレクトリ部348とスナップショット部350の2つに分かれる。ディレクトリ部348はターゲット命令識別子(例えば、各ターゲット命令の絶対または相対アドレス)を格納するのに用いられ、スナップショット部350は、これらのターゲット命令識別子(target instruction identifier)に関連する仮想スタック344のスナップショットを格納するのに用いられる。“ターゲット”命令は、ジャンプまたは分岐命令の行き

先となり得るバイトコード命令として定義される。例えば、条件付き分岐命令は、条件(満たされることも満たされないこともある)と、条件が成立したとき、処理が“ジャンプ”すべきプログラム内の場所(ターゲット)を示す分岐先支持部を含む。条件付きジャンプ命令を評価する際、ペリファイヤ240は、スタックスナップショット格納構造346を用いて、ジャンプする直前の、ターゲット命令識別子をディレクトリ部348内に格納するとともに、仮想スタック344のステータスをスナップショット部350内に格納する。スタックスナップショット格納構造346の動作については、バイトコードペリファイヤプログラムの動作と共に、後により詳細に説明する。

【0054】上述したように、バイトコードプログラム340は、データタイプに依存する複数の命令を含んでおり、それらの命令の各々は、本発明のペリファイヤ240によって評価される。バイトコードプログラム350は、スタック操作に関する命令352(スタックへの整数のプッシュ)及び354(スタックからの整数のポップ)、前方ジャンプ356とその関連するターゲット364、後方ジャンプ366とその関連するターゲット362、及びDOループ358とその終わり(end)360(DOループのタイプによって、条件付きまたは無条件分岐命令であってもよい)を含んでいる。本発明の好適実施例のペリファイヤ240は、スタック操作及びデータタイプコンパチビリティをペリファイヤするだけなので、このバイトコードペリファイヤの動作は、これらの代表的な命令セットを用いることによって説明することができる。

【0055】図6乃至図12、及び付録2を参照して、バイトコードペリファイヤプログラム240の動作について以下に詳細に説明する。付録2は、ペリファイヤプログラムを疑似コードで表したものである。付録2の疑似コードは、本質的には、従来の汎用コンピュータ言語を用いたコンピュータ言語と同等である。この疑似コードは、本発明の説明のためにのみ作られたものであって、本分野の通常の知識を有するコンピュータプログラマであれば容易に理解されるように意図して書かれている。

【0056】図6に示されているように、ダウンロードされたバイトコードプログラムは、処理のためバイトコードペリファイヤ240にロードされる(400)。ペリファイヤ240は、オペランド及びローカル変数のデータタイプ情報を格納するため、メモリ内にアレイ状に位置を指定することによって、仮想スタック344及び仮想ローカル変数アレイ345を生成する(402)。同様に、ペリファイヤは、スナップショット情報を格納するため、メモリ内にアレイ上に位置を指定することによってスタックスナップショット格納構造を生成する(404)。最後に、ペリファイヤは、仮想スタックの

エントリ数を追跡するためのスタックカウンタ342として働くレジスタを指定する(406)。

【0057】第1パスは、条件付き及び無条件ジャンプ及びループ命令に関連するターゲット情報を抽出するべく、バイトコードプログラム内に形成されたものである。第1パスに於いて、ペリファイヤ300は全ての命令を順次処理していき(ステップ408、410、412)、命令が条件付きまたは無条件ジャンプ命令の場合(ステップ414)、かつそのジャンプに対するターゲット位置がまだディレクトリ348内に記憶されていない場合(ステップ418)、ターゲット位置を示す情報をスタックスナップショット格納構造346のディレクトリ部348に格納する(ステップ416)。例えば、ターゲット命令の絶対または相対アドレスを、ディレクトリ部348の次に使用可能なスロットに格納しても良い。他のタイプのバイトコード命令は、この第1パスでは無視される。

【0058】プログラム内の全ての命令に対し処理が終了した後、ディレクトリ部348をソートして、リストアップされたターゲット位置をアドレス順に並び替えることが好ましい(420)。

【0059】再度図5を参照されたい。図5では、説明のため、スタックスナップショット格納構造346は、既にペリフィケーションの第1パスがバイトコードプログラム350内に示されているバイトコード命令に基づいて終了したように、ディレクトリ部348内に情報が格納され、ロードされている。即ち、ディレクトリ部には、バイトコードプログラム内にある条件付き及び無条件ジャンプ命令の全てのターゲットのアドレスがロードされている。

【0060】図7では、バイトコードプログラムによるオペランドスタック及びデータタイプの使用が適切かどうかをペリファイするため、バイトコードプログラムの第2のパスが開始される。バイトコードプログラムの最初の命令が選択されると(430)、ペリファイヤは、選択された命令に対するアドレスが、上述した第1パスに於いて既にスタックスナップショット格納構造346のディレクトリ部348内に格納されているかどうかチェックする(432)。

【0061】選択された命令のアドレスがディレクトリ348内にある場合(即ち、選択された命令が条件付きまたは無条件ジャンプのターゲットであることを意味している)、ペリファイヤは関連するスタックスナップショットがスタックスナップショット格納構造346のスナップショット部350内に格納されているかどうかをチェックする(434)。スタックスナップショットが格納されていない場合(即ち、命令が後方ジャンプのターゲットであることを意味する)、スタックカウンタ及び仮想スタックの内容が、スタックスナップショット格納構造346内に格納される(436)。スナップシ

ョットは、処理中の命令を実行する直前の仮想スタックのステータスに関する情報を含み、それにはスタックにプッシュされている各データに対するデータタイプ値も含まれる。ペリファイヤは、その後ペリフィケーションプロセスを続け、後に述べるように、ステップ450からは個々の命令に対し解析を開始する。

【0062】スタックスナップショットが、選択されている命令に対し格納されている場合(このターゲット命令に関連するジャンプ命令が既に処理されているということの意味する)、ペリファイヤは、スタックスナップショット格納構造346のスナップショット部350内に格納されている、この命令に対する仮想スタックスナップショット情報と、仮想スタックのそのときの状態とを比較する(438)。比較の結果、仮想スタックの状態とスナップショットとがマッチしないことが示された場合(スタックスステータスマISMATCH)、スタックスステータスマISMATCHが発生したバイトコードプログラム内の場所を示すエラーメッセージまたは信号が生成される(440)。好適実施例では、仮想スタックとスナップショットが同数または同タイプのエントリを含んでいない場合にミスマッチが発生する。ペリファイヤは、その後、そのプログラムが“偽(False)”であることを示すようにペリフィケーションステータス245をセットし、ペリフィケーションプロセスをアボートする(442)。プログラムが“偽”であることを示すようにペリフィケーションステータス245をセットすることにより、バイトコードインタープリタ242によるプログラムの実行が阻止される(図3)。

【0063】選択されている命令に対し既に格納されているスタックスナップショットと、その時の仮想スタックの状態とがマッチする場合(438)、ペリファイヤはペリフィケーションプロセスを続け、後に述べるようにステップ450から個々の命令に対して分析を開始する。

【0064】選択されている命令のアドレスがスタックスナップショット格納構造346のディレクトリ部348内に見つからない場合、またはスタックスステータスマISMATCHが検出されない場合、ペリファイヤは、その命令特有のスタック使用法及び機能に応じて、命令に対する一連のチェックのうち、いくつかを選択して実行する。

【0065】図8を参照されたい。最初に行われるチェックは、オペランドスタックからデータをポップさせる命令に関する。選択されている命令によってデータがスタックからポップされる場合(450)、スタックカウンタが調べられ、命令のデータポップ要求を満足する十分なデータがスタック内にあるかどうかが判定される(452)。

【0066】オペランドスタックがその命令に対し十分なデータを持っていない場合(452)、これはスタ

クアンダーフローと呼ばれ、スタックアンダーフローが検出されたプログラム内の場所を示すエラー信号またはメッセージが生成される(454)。更にペリファイヤはプログラムが“偽”であることを示すようにペリフィケーションステータス245をセットし、ペリフィケーションプロセスをアボートする(456)。

【0067】スタックアンダーフロー状態が検出されない場合、ペリファイヤは、予め仮想スタック内に格納されているデータタイプコード情報と、(もしあれば)選択されている命令のデータタイプ要求とを比較する。例えば、分析されている命令の操作コード(opcode)がスタックからポップされた値の整数型の加算(integer add)を要求する場合、ペリファイヤは、ポップされている仮想スタックのアイテムのオペランド情報を比較して、それが適切なデータタイプであること、即ち整数であることを確認する。比較がマッチする結果となった場合、ペリファイヤは、ポップされているエントリに関連する情報を仮想スタックから削除し、仮想スタック344からポップされたエントリ数が反映されるようにスタックカウンタ342を更新する。

【0068】仮想スタック344のポップされているエントリのオペランド情報と、選択されている命令のデータタイプ要求との間でミスマッチが検出された場合(458)は、ミスマッチがバイトコードプログラム内のどこで発生したかを示すメッセージが生成される(462)。更に、ペリファイヤは、プログラムが“偽”であることを示すようにペリフィケーションステータス245をセットし、ペリフィケーションプロセスをアボートする(456)。これで、ポップペリフィケーションプロセスが終了する。

【0069】次に、図9を参照されたい。選択されている命令がデータをスタックにブッシュしようとする場合(470)、スタックカウンタが調べられ、その命令がブッシュしようとするデータを格納するための十分な余地がスタックにあるかどうかが判定される(472)。オペランドスタックに、その命令がブッシュしようとするデータを格納するための十分な余地がない場合(472)、これはスタックオーバフローと呼ばれ、スタックオーバフローが検出されたプログラム内の場所を示すエラー信号またはメッセージが生成される(474)。更に、ペリファイヤは、プログラムが“偽”であることを示すようにペリフィケーションステータス245をセットし、ペリフィケーションプロセスをアボートする(476)。

【0070】スタックオーバフロー状態が検出されない場合、ペリファイヤは、選択されている命令によってスタックにブッシュされる各データに対し、(実際にプログラムが実行される際に)オペランドスタックにブッシュされるデータ(オペランド)のタイプを示すエントリを仮想スタックに加える(478)。この情報は、本発

明の好適実施例のバイトコードプログラムで用いられている、データタイプが特定された操作コードから導かれる。また、ペリファイヤは、スタックカウンタ342を更新して、仮想スタックに追加された1または複数のエントリが反映されるようにする。これでスタックブッシュペリフィケーションプロセスが終了する。

【0071】図10を参照されたい。選択されている命令により、通常の逐次的ステップを飛び越すような、条件付きまたは無条件の前方ジャンプまたは分岐が発生する場合(480)、ペリファイヤは、まずそのジャンプ命令のターゲット位置に対するスナップショットがスタックスナップショット格納構造346内に格納されているかどうかをチェックする。スタックスナップショットが格納されていない場合、(そのジャンプに関連する仮想スタックの更新がなされた後の)仮想スタック状態が、スタックスナップショット格納構造346内の、そのターゲットプログラム位置に関連する位置に格納される(484)。このジャンプに関連するスタックポップ操作は、既に実行されているステップ460に於いて仮想スタック内に反映されていることに注意されたい(図8参照)。

【0072】スタックスナップショットが格納されている場合(このターゲットに関連する他のエントリポイント(entry point)が既に処理されていることを示す)、ペリファイヤは、スタックスナップショット格納構造346のスナップショット部350内に格納されている仮想スタックスナップショット情報と、仮想スタックのその時の状態とを比較する。比較の結果、スナップショットと仮想スタックの状態とがマッチしていないことが示された場合(スタックスステータスミスマッチ)、スタックスステータスミスマッチが発生したバイトコードプログラム内の場所を示すエラーメッセージが生成される(488)。好適実施例では、スナップショットとその時の仮想スタックが、同数または同タイプのエントリを含んでいないときミスマッチが発生する。更に、その時の仮想スタック内の1以上のデータタイプ値が、スナップショット内の対応するデータタイプ値に一致しないときにもミスマッチが発生する。ペリファイヤは、その後プログラムが“偽”であることを示すようにペリフィケーションステータス245をセットし、ペリフィケーションプロセスをアボートする(490)。ステップ486でスタックスステータスマッチングが検出された場合、ペリファイヤはステップ500へと処理を継続する(図11)。

【0073】図11を参照されたい。選択されている命令によってプログラムの後方に向かう条件付きまたは無条件ジャンプまたは分岐が発生する場合(ステップ500)、ペリファイヤは、スタックスナップショット格納構造346のスナップショット部350内に格納されている、その後方ジャンプのターゲットに関連する仮想ス

タックスナップショット情報(ステップ436で格納した)と、仮想スタックのその時の状態とを比較する。比較の結果、スナップショットとその時のステータスがマッチしないことが示された場合(スタックステータスマISMATCH)、スタックステータスマISMATCHが発生したバイトコードプログラム内の場所を示すエラーメッセージが生成される(504)。好適実施例では、MISMATCHは、その時の仮想スタックとスナップショットが同数または同タイプのエントリを含んでいないか、あるいは仮想スタック内のデータタイプエントリのどれかが、スナップショット内の対応するデータタイプエントリとマッチしない場合に発生する。ペリファイヤは、その後プログラムが“偽”であることを示すようにペリフィケーションステータス245をセットし、ペリフィケーションプロセスをアボートする(506)。

【0074】スタックステータスがマッチしていることが検出されるか(ステップ502)、または命令が後方ジャンプでない場合(ステップ500)、ペリファイヤはステップ510へと処理を継続する。

【0075】選択されている命令が、ローカル変数からデータを読み出す場合(510)、ペリファイヤは対応する仮想ローカル変数内に予め格納されているデータタイプコード情報と、(もしあれば)選択されている命令のデータタイプ要求とを比較する。仮想ローカル変数に格納されているデータタイプ情報と選択されている命令のデータタイプ要求との間にMISMATCHが検出された場合(512)、MISMATCHが発生したバイトコードプログラム内の場所を示すメッセージが生成される(514)。その後、ペリファイヤはプログラムが“偽”であることを示すようにペリフィケーションステータス245をセットし、ペリフィケーションプロセスをアボートする(516)。

【0076】その時選択されている命令がローカル変数からデータを読み出さない場合(510)、またはステップ512に於けるデータタイプの比較がマッチする結果となった場合、ペリファイヤは、選択されている命令の処理を更にステップ520へと継続する。

【0077】図12を参照されたい。選択されている命令によってデータがローカル変数に格納される場合(520)、対応する仮想ローカル変数が調べられ、その仮想ローカル変数がデータタイプ値を格納しているかどうか判定される(522)。その仮想ローカル変数がデータタイプ値を格納している場合(即ちそのデータは以前にそのローカル変数内に格納されたことを意味する)、ペリファイヤは、その仮想ローカル変数内にあるデータタイプ情報と、選択されているバイトコード命令に関するデータタイプとを比較する(524)。仮想ローカル変数内に格納されているデータタイプ情報と、選択されている命令によって要求されているデータタイプとの間にMISMATCHが検出されると(524)、MISMATCH

が発生したバイトコードプログラム内の場所を示すメッセージが生成される(526)。その後、ペリファイヤはプログラムが“偽”であることを示すようにペリフィケーションステータス245をセットし、ペリフィケーションプロセスをアボートする(528)。

【0078】選択されている命令が、ローカル変数にデータを格納しない場合(520)、選択されている命令に対する処理は終了する。選択されている命令によってデータがローカル変数内に格納されるが、その仮想ローカル変数がデータタイプ値を格納していない場合(即ち、そのローカル変数にデータを格納するような命令がペリファイヤによってまだ処理されていないことを示す)、選択されているバイトコード命令に関連するデータタイプが、その仮想ローカル変数に格納される(ステップ530)。

【0079】続いて、ペリファイヤは、その命令が処理中のバイトコードプログラム340の最後の命令であるかどうかをチェックする(540)。処理すべき命令がまだ残っている場合、ペリファイヤは次の命令をロードし(542)、ペリフィケーションプロセスをステップ432から繰り返す。それ以上処理する命令がない場合は、ペリファイヤはプログラムが“真”であることを示すようにペリフィケーションステータス245をセットし(544)、ペリフィケーションプロセスの終了を伝える。

【0080】バイトコードインタープリタ

図13のフローチャート及び付録3を参照しつつ、バイトコードインタープリタ242の動作について説明する。付録3は、バイトコードインタープリタを疑似コードで表したものである。

【0081】指定されたバイトコードプログラムが与えられた場合、即ち、実行すべきプログラムとして選択された後(560)、バイトコードプログラムインタープリタ242は、指定されたバイトコードプログラムの健全性をペリファイヤするべく、バイトコードペリファイヤ240を呼び出す(562)。バイトコードペリファイヤについては既に上述した。

【0082】ペリファイヤが“ペリフィケーション失敗(verification failure)”をリターンしてきた場合(564)、指定されたバイトコードプログラムのインタープリタによって実行されることなくアボートされる(566)。

【0083】ペリファイヤ242が“ペリフィケーション成功”をリターンしてきた場合(564)、指定されたバイトコードプログラムはリソースユーティリティプログラムにリンクされ(568)、更に、そのプログラムによって参照される他のプログラム、関数、及びオブジェクトにもリンクされる。そのようなリンク過程は多くのプログラムインタープリタに於ける従来の前処理(pre-execution)過程と同様である。その後リンクさ

れたバイトコードプログラムはインタープリタによってインタープリットされ実行される(570)。本発明のバイトコードインタープリタは、プログラムの実行中、オペランドスタックオーバーフロー及びアンダーフローのチェックをしない。また、オペランドスタック内に格納されているデータに対するデータタイプのチェックもしない。このような従来インタープリット過程で行われていたスタックオーバーフロー／アンダーフロー、及びデータタイプのチェックは、本発明では省略することができる。これは、ペリファイヤによって、このようなエラーがプログラム実行中に発生しないことが既にペリファイ

【0084】本発明のプログラムインタープリタは、何度も繰り返し実行される命令ループ(instruction loop)を有するバイトコードプログラムの実行に対し、特に効果的である。なぜなら、本発明では、そのような各命令ループ内の各バイトコードに対して、オペランドスタックチェック命令は一度だけしか行われないからである。対照的に、従来のインタープリタによるプログラムの実行に於いては、インタープリタは、オーバーフロー(即ち、スタックが格納できるより多くのデータがスタックに加えられる)やアンダーフロー(即ち、スタックが空の時にスタックからデータをポップしようとする)が生じていないかどうか、オペランドスタックを連続的*

＊にモニタする必要がある。そのようなスタックモニタリングは、通常、スタックのステータスを変化させる全ての命令(即ち、ほとんど全ての命令)に対して実行しなければならない。多くのプログラムで、インタープリタによって実行されるスタックモニタリング命令は、インタープリットされるコンピュータプログラムの実行時間の約80%に達する。その結果、本発明のインタープリタは、しばしば、同じコンピュータ上で実行される従来のプログラムインタープリタの2倍から5倍の速さでプログラムを実行する。

【0085】本発明の特定の実施例について行った上述の説明は、例示を目的としたものであり、本発明を網羅的に記載することを意図したものでも、本発明を開示した形態そのものに限定することを意図したものでもなく、上述したことから多くの変形変更が可能であることは明らかであろう。上記の実施例は、本発明の原理とその実際の応用が最も良く説明されるように選択及び記述したものであり、それによって当業者が本発明を最も良く利用して、特定の使用に合うように様々な改良を加えた変形実施態様が可能なようにしたものである。本発明の範囲は、特許請求の範囲によって画定される。

【0086】

【実施例】

表1

OAK言語のバイトコード

| 命令の名称 | 簡単な説明 |
|---------------|-------------------------------------|
| aaload | アレイからのオブジェクトリファレンスのロード |
| aastore | オブジェクトリファレンスのオブジェクトリファレンスア |
| レ | |
| | イへの格納 |
| aconst_null | ヌルオブジェクト(null object)のプッシュ |
| aload | ローカルオブジェクト変数のロード |
| areturn | 関数からのオブジェクトリファレンスのリターン |
| arraylength | アレイ長の取得 |
| astore | オブジェクトリファレンスのローカル変数への格納 |
| astore_<n> | オブジェクトリファレンスのローカル変数への格納 |
| athrow | 例外の破棄 |
| bipush | 1バイト符号付整数のプッシュ |
| breakpoint | ブレイクポイントハンドラ(breakpoint handler)のコー |
| ル | |
| catchsetup | 例外ハンドラ(exception handler)のセットアップ |
| catchteardown | 例外ハンドラのリセット |
| checkcast | オブジェクトが所与のタイプであることの確認 |
| df2 | 倍精度浮動小数点数の単精度浮動少数点数への変換 |
| d2i | 倍精度浮動小数点数の整数への変換 |
| d2l | 倍精度浮動小数点数の倍長整数への変換 |
| dadd | 倍精度浮動小数点数の加算 |
| daload | 倍精度浮動小数点数のアレイからのロード |
| dastore | 倍精度浮動小数点数のアレイへの格納 |

| | |
|------------|-------------------------------|
| 27 | 28 |
| dcmpq | 2つの倍精度浮動小数点数の比較（比較できない場合 1 を |
| リ | ターン） |
| dcmpl | 2つの倍精度浮動小数点数の比較（比較できない場合 - 1 |
| を | リターン） |
| dconst_cdb | 倍精度浮動小数点数のプッシュ |
| ddiv | 倍精度浮動小数点数の除算 |
| dload | 倍精度浮動小数点数のローカル変数からのロード |
| dload_cnb | 倍精度浮動小数点数のローカル変数からのロード |
| dmod | 倍精度浮動小数点数のモジュール関数の実行 |
| dmul | 倍精度浮動小数点数の乗算 |
| dneg | 倍精度浮動小数点数の無効化 |
| dreturn | 倍精度浮動小数点数の関数からのリターン |
| dstore | 倍精度浮動小数点数のローカル変数への格納 |
| dstore_cnb | 倍精度浮動小数点数のローカル変数への格納 |
| dsub | 倍精度浮動小数点数の引き算 |
| dup | 一番上のスタックワードの複製 |
| dup2 | 一番上の2つのスタックワードの複製 |
| dup2_x1 | 一番上の2つのスタックワードを複製し2つ下げる |
| dup2_x2 | 一番上の2つのスタックワードを複製し3つ下げる |
| dup_x1 | 一番上のスタックワードを複製し2つ下げる |
| dup_x2 | 一番上のスタックワードを複製し3つ下げる |
| f2d | 単精度浮動小数点数の倍精度浮動小数点数への変換 |
| f2i | 単精度浮動小数点数の整数への変換 |
| f2l | 単精度浮動小数点数の倍長整数への変換 |
| fadd | 単精度浮動小数点数の加算 |
| faload | 単精度浮動小数点数のアレイからのロード |
| fastore | 単精度浮動小数点数のアレイへの格納 |
| fcmpq | 単精度浮動小数点数の比較（比較不能な場合 1 を返す） |
| fcmpl | 単精度浮動小数点数の比較（比較不能な場合 - 1 を返す） |
| fconst_cf | 単精度浮動小数点数のプッシュ |
| fdiv | 単精度浮動小数点数の除算 |
| fload | 単精度浮動小数点数のローカル変数からのロード |
| fload_cnb | 単精度浮動小数点数のローカル変数からのロード |
| fmod | 単精度浮動小数点数のモジュール関数の実行 |
| fmul | 単精度浮動小数点数の乗算 |
| fneg | 単精度浮動小数点数の無効化 |
| freturn | 単精度浮動小数点数の関数からのリターン |
| fstore | 単精度浮動小数点数のローカル変数への格納 |
| fstore_cnb | 単精度浮動小数点数のローカル変数への格納 |
| fsub | 単精度浮動小数点数の引き算 |
| getfield | オブジェクトからのフィールドのフェッチ |
| getstatic | クラスからのスタティックフィールドのセット |
| goto | 無条件分岐 |
| i2d | 整数の倍精度浮動小数点数への変換 |
| i2f | 整数の単精度浮動小数点数への変換 |
| i2l | 整数の倍長整数への変換 |
| iadd | 整数の加算 |
| iaload | 整数のアレイからのロード |

| | |
|-----------------|-------------------------------------|
| iand | 2つの整数のブール代数AND |
| iastore | 整数のアレイへの格納 |
| iconst_<n> | 整数のプッシュ |
| iconst_m1 | 整数定数から1を引いてプッシュ |
| idiv | 整数の除算 |
| if_acmpeq | オブジェクトが同じ場合分枝 |
| if_acmpne | オブジェクトが同じでない場合分枝 |
| if_icmpeq | 整数が等しい場合分枝 |
| if_icmpge | 整数がある値以上の場合分枝 |
| if_icmpgt | 整数がある値より大きい場合分枝 |
| if_icmple | 整数がある値以下の場合分枝 |
| if_icmplt | 整数がある値未満の場合分枝 |
| if_icmpne | 整数がある値に等しくない場合分枝 |
| ifeq | 0に等しい場合分枝 |
| ifge | 0以上の場合分枝 |
| ifgt | 0より大きい場合分枝 |
| ifle | 0以下の場合分枝 |
| iflt | 0未満の場合分枝 |
| ifne | 0に等しくない場合分枝 |
| iinc | 定数分だけローカル変数を増加 |
| iload | 整数のローカル変数からのロード |
| iload_<n> | 整数のローカル変数からのロード |
| imod | 整数のモジュール関数の実行 |
| imul | 整数の乗算 |
| ineq | 整数の無効化 |
| instanceof | オブジェクトが所与のタイプであるかどうかの判定 |
| int2byte | 整数の符号付バイト (signed byte) への変換 |
| int2char | 整数のキャラクタ (char) への変換 |
| invokeinterface | インタフェースメソッドの呼び出し |
| invokemethod | クラスメソッドの呼び出し |
| invokesuper | スーパークラスメソッドの呼び出し |
| ior | 2つの整数のブール代数OR |
| ireturn | 整数の関数からのリターン |
| ishl | 整数の左シフト |
| ishr | 整数の算術的右シフト (arithmetic shift right) |
| istore | 整数のローカル変数vindexへの格納 |
| istore_<n> | 整数のローカル変数nへの格納 |
| isub | 整数の引き算 |
| iushr | 整数の論理右シフト |
| ixor | 2つの整数のブール代数XOR |
| jsr | サブルーチンへのジャンプ |
| l2d | 倍長整数の倍精度浮動小数点数への変換 |
| l2f | 倍長整数の単精度浮動小数点数への変換 |
| l2i | 倍長整数の整数への変換 |
| ladd | 倍長整数の加算 |
| laload | 倍長整数のアレイからのロード |
| land | 2つの倍長整数のブール代数AND |
| lastore | 倍長整数のアレイへの格納 |
| lcmp | 倍長整数の比較 |
| lconst_<l> | 倍長整数定数のプッシュ |

| | | |
|--------------|---|----|
| 31 | | 32 |
| ldc1 | 定数ブールからのアイテムのプッシュ | |
| ldc2 | 定数ブールからのアイテムのプッシュ | |
| ldc2w | 定数ブールからのロングまたはダブルのプッシュ | |
| ldiv | 倍長整数の除算 | |
| lload | 倍長整数のローカル変数からのロード | |
| lload_<n> | 倍長整数のローカル変数からのロード | |
| lmod | 倍長整数のモジュロ関数の実行 | |
| lmul | 倍長整数の乗算 | |
| lneg | 倍長整数の無効化 | |
| lookswitch | キーマッチ (key match) によるジャンプテーブルへのアクセス及びジャンプ | |
| lor | 2つの倍長整数のブール代数OR | |
| lreturn | 倍長整数の関数からのリターン | |
| lshl | 倍長整数の左シフト | |
| lshr | 倍長整数の算術的右シフト | |
| lstore | 倍長整数のローカル変数への格納 | |
| lstore_<n> | 倍長整数のローカル変数への格納 | |
| lsub | 倍長整数の引き算 | |
| lushr | 倍長整数の論理右シフト | |
| lxor | 倍長整数のブール代数XOR | |
| monitorenter | コードのモニタされている領域へのエンター | |
| monitorexit | コードのモニタされている領域からの離脱 | |
| new | 新たなオブジェクトの生成 | |
| newarray | 新たなアレイの位置決め | |
| newfromname | ネーム (name) からの新たなオブジェクトの生成 | |
| nop | 何もしない | |
| pop | 1 番上のスタックワードのポップ | |
| pop2 | 1 番上の2つのスタックワードのポップ | |
| putfield | オブジェクト内のフィールドのセット | |
| putstatic | クラス内のスタティックフィールドのセット | |
| ret | サブルーチンからのリターン | |
| return | プロシージャ (procedure) からのリターン (ボイド) | |
| saload | 符号付バイトのアレイからのロード | |
| sastore | 符号付バイトのアレイへの格納 | |
| siaload | 符号なしショート (short) のアレイからのロード | |
| siastore | 符号なしショートのアレイへの格納 | |
| sipush | 2 バイト符号付整数のプッシュ | |
| tableswitch | インデックスによるジャンプテーブルへのアクセス及びジャンプ | |
| verifystack | スタックが空であることをベリファイする | |

【0087】

付録1

クラスローダの疑似コード

見たいオブジェクト (“被参照オブジェクト”) のユーザによる選択 (例えば、ある文献内のオブジェクトまたは他のオブジェクトへのハイパーリンクを選択することによって選択することが出来る)

被参照オブジェクトを格納しているサーバへの接続を開く

被参照オブジェクトに対するハンドル (データタイプを含む) の受け取り

データタイプがユーザシステムにとって既知であるかどうか (即ち、受け取ったデータタイプのオブジェクトに対応するビューをユーザが持っているかどうか)

) のチェック

データタイプが既知でない場合

```
{
  同じサーバへの第 2 の接続を開く
  指定されたデータタイプに対応するビューワのリクエスト：
  成功の場合
    /*ハイブリッドシステムオプション：ノンバイトコードビューワ
    (non-bytecode viewer) に対するチェック*/
    {
      受け取ったビューワがバイトコードプログラムでない場合
      {
        ビューワをアクセプトするか否かの決定
        ビューワがアクセプトされない場合
          受け取ったビューワの削除
        ビューワがアクセプトされる場合 /*ノンバイトコードビューワ
        がアクセプトされる*/
          FinishObjectDownloadへ行く
        }
      /*ベリフィケーション及びレジストレーションプロシージャ*/
      受け取ったビューワがバイトコードプログラムの場合
      {
        受け取ったビューワに対するバイトコードベリファイヤの実行
        ベリフィケーション成功の場合
        {
          /*ビューワのレジストレーション*/
          受け取ったビューワをベリファイ済みとしてマーキングする
          ビューワのローカルビューワライブラリへの格納
          既知データタイプリストに、そのデータタイプを加える
        }
        ベリフィケーションが不成功の場合
          受け取ったビューワの削除
        }
      }
    }
  データタイプがなお認識されない場合
  {
    指定されたデータタイプに対応するビューワを他のサーバ中でサーチ
    成功の場合
      {上記と同じベリフィケーション及びレジストレーションプロシージャ
      の実行}
    }
  データタイプがなお認識されない場合
  {
    被参照オブジェクトのダウンロードのアボート
    被参照オブジェクトに対応するビューワを見つけることができなかった
    ことをユーザに通知
    }
}
```

FinishObjectDownload: /*ノンバイトコードビューワに対する分岐先*/
 被参照オブジェクトのダウンロードの完了
 ダウンロードされたオブジェクトに組み込みバイトコードプログラムが含まれて

いる場合

```
{
  組み込みプログラムに対するバイトコードベリファイヤの実行
  ベリフィケーション成功の場合
    組み込みプログラムをベリファイ済みとしてマーキングする
  ベリフィケーション不成功の場合
    {
      受け取ったオブジェクトの削除
      ダウンロードプロセスのアボート
    }
}
```

関連するデータタイプに対応するビューワを用いて被参照オブジェクトを見る

【0088】

付録2

OAKバイトコードベリファイヤの疑似コード

```
ベリファイするバイトコードプログラムの受け取り
スタックステータス情報を格納するための仮想オペランドスタックデータ構造の
生成及びローカル変数データタイプ情報を格納するための仮想ローカル変数アレ
イの生成
仮想スタックスナップショットを格納するためのデータ構造の生成
バイトコードプログラムを通る第1パス：
  条件付き及び無条件ジャンプまたは分岐のターゲットとなる（即ち、2以上
  の先行命令からエンターされ得る）命令を全て見つける
  仮想スタックスナップショットデータ構造にそれらのターゲット命令のリス
  トを格納する
バイトコードプログラムを通る第2パス：
  VerificationSuccessを“真”にセットする
  最後のバイトコード命令を処理するまで繰り返す（DOループの開始）：
  {
    次のバイトコード命令を（プログラム内の順番通りに）選択する
    命令がターゲット命令のリスト内にある場合
    {
      この命令に対する仮想スタックのナップショットが既に存在する
      場合
      {
        仮想スタックの現在のステータスを格納されているナップシ
        ョットと比較する
        スナップショットが現在の仮想スタックステータスとマッチし
        ない場合
        {
          スタックミスマッチが発生したプログラム内の場所を示す
          メッセージをプリントする
          ベリフィケーションをアボートする
          VerificationSuccessを“偽”にセットする
          リターン
        }
      }
    }
    この命令に対する仮想スタックのナップショットが存在しない場
    合
    現在の仮想スタックステータスのナップショットを格納する
```

}

命令のタイプによる場合分け：

{

命令によってデータがオペランドスタックからポップされる場合

{

スタックアンダーフローのチェック

スタックアンダーフローの場合

{

アンダーフローが発生したプログラム内の場所を示すメッセージをプリントする

ベリフィケーションをアボートする

リターン

}

スタックからポップされる各オペランドのデータタイプをバイトコード命令によって要求されているデータタイプと比較するタイプミスマッチの場合

{

データタイプミスマッチが発生したプログラム内の場所を示すメッセージをプリントする

VerificationSuccessを“偽”にセットする

}

ポップされるオペランドに対する情報を仮想スタックから削除する

スタックカウンタを更新する

}

命令によってデータがオペランドスタックにプッシュされる場合

{

スタックオーバフローのチェック

スタックオーバフローの場合

{

オーバフローが発生したプログラム内の場所を示すメッセージをプリントする

ベリフィケーションをアボートする

VerificationSuccessを“偽”にセットする

リターン

}

オペランドスタックにプッシュされるデータのデータタイプを示す情報を仮想スタックに加える

スタックカウンタを更新する

}

命令が前方ジャンプまたは分岐命令の場合

{

そのターゲット命令に対する仮想スタックのスナップショットが既に存在する場合

{

仮想スタックの現在のステータスを格納されているスナップショットと比較する

スナップショットが現在の仮想スタックステータスとマッチしない場合

{

```

        スタックミスマッチが発生したプログラム内の場所を
        示すメッセージをプリントする
        ベリフィケーションをアボートする
        VerificationSuccessを“偽”にセットする
        リターン
    }
}

```

そのターゲット命令に対する仮想スタックのスナップショット
が存在しない場合

現在の仮想スタックステータスのスナップショットをその
ターゲット命令に対するスナップショットとして格納する

```

}

```

命令がループの終わりの後方ジャンプ命令或いは他の後方ジャンプ
または分岐命令の場合

```

{
    現在のスタックステータスをターゲット命令に対して格納され
    ているスナップショットと比較する
    現在の仮想スタックステータスが格納されているスナップシ
    ョットとマッチしない場合
    {
        スタックミスマッチが発生したプログラム内の場所を示す
        メッセージをプリントする
        ベリフィケーションをアボートする
        VerificationSuccessを“偽”にセットする
        リターン
    }
}

```

命令によってローカル変数からデータが読み出される場合

```

{
    ローカル変数から読み出される各データタイプをバイトコード
    命令によって必要とされるデータタイプと比較する
    タイプミスマッチの場合
    {
        データタイプミスマッチが発生したプログラム内の場所を
        示すメッセージをプリントする
        VerificationSuccessを“偽”にセットする
    }
}

```

命令によってローカル変数内にデータが格納される場合

```

{
    対応する仮想ローカル変数に既にデータタイプ値が格納されて
    いる場合
    {
        仮想ローカル変数に格納されているデータタイプ値を対応
        するローカル変数に格納されるデータのデータタイプ（現
        在のバイトコード命令によって扱われるデータタイプによ
        って決定される）と比較する
        タイプミスマッチの場合
        {

```

41

42

```

        データタイプミスマッチが発生したプログラム内の場
        所を示すメッセージをプリントする
        VerificationSuccessを“偽”にセットする
    }
}
対応する仮想ローカル変数にデータタイプ値が格納されてい
ない場合
    対応するローカル変数に格納されるデータのデータタイプ
    を示す情報を仮想ローカル変数に加える
}
} /* 場合分けの終わり */
} /* DOループの終了 */
リターン (ベリフィケーション成功)

```

【0089】

付録3

バイトコードインタープリタの疑似コード

```

実行すべき、指定されたバイトコードプログラムの受け取り
指定されたバイトコードプログラムをベリファイするためのバイトコードベリフ
ァイアのコール
ベリフィケーション成功の場合
{
    指定されたバイトコードプログラムをリソースのユーティリティプログラム
    にリンクする
    指定されたバイトコードプログラムの命令を、オペランドスタックオーバフ
    ロー及びアンダーフローに対するチェックを行うことなく、またオペランド
    スタックに格納されているオペランドに対するデータタイプのチェックを行
    うことなくインタープリットし、実行する
}

```

【図面の簡単な説明】

【図1】図1は、ネットワークを介して接続された2つ 30 のコンピュータを示している。

【図2】図2は、ネットワークを介して接続された2つ のコンピュータを示しており、それらのコンピュータの うち少なくとも一つは、様々な実行可能形式のソースプ ログラムの複数のコピーを格納するための補助記憶装置 (ディスク記憶装置)を含んでいる。

【図3】図3は、ネットワークを介して接続された2つ のコンピュータを示しており、それらのコンピュータの うち少なくとも一つは、本発明によるバイトコードプ ログラムベリファイとクラスローダを含んでいる。

【図4】図4は、本発明の好適実施例に従って、リモートサーバ内にあるビューワとバイトコードプログラムに アクセスするためのロードプロセスのフローチャートで ある。

【図5】図5は、本発明によるバイトコードプログラムのベリフィケーションに於いてバイトコードベリファイヤによって維持されるデータ構造を示している。

【図6】図6は、本発明の好適実施例に於けるバイトコードプログラムベリフィケーションプロセスのフローチャートの一部である。

【図7】図7は、本発明の好適実施例に於けるバイトコードプログラムベリフィケーションプロセスのフローチャートの一部である。

【図8】図8は、本発明の好適実施例に於けるバイトコードプログラムベリフィケーションプロセスのフローチャートの一部である。

【図9】図9は、本発明の好適実施例に於けるバイトコードプログラムベリフィケーションプロセスのフローチャートの一部である。

【図10】図10は、本発明の好適実施例に於けるバイトコードプログラムベリフィケーションプロセスのフローチャートの一部である。

【図11】図11は、本発明の好適実施例に於けるバイトコードプログラムベリフィケーションプロセスのフローチャートの一部である。

【図12】図12は、本発明の好適実施例に於けるバイトコードプログラムベリフィケーションプロセスのフローチャートの一部である。

【図13】図13は、本発明の好適実施例に於けるバイトコードプログラムインタープリットプロセスのフローチャートである。

50 【符号の説明】

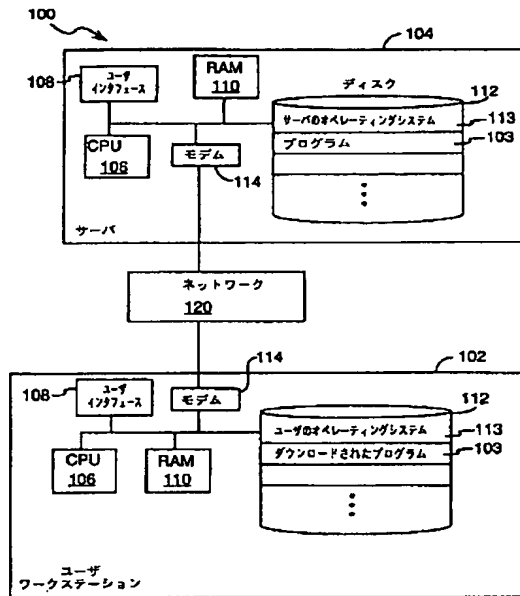
43

44

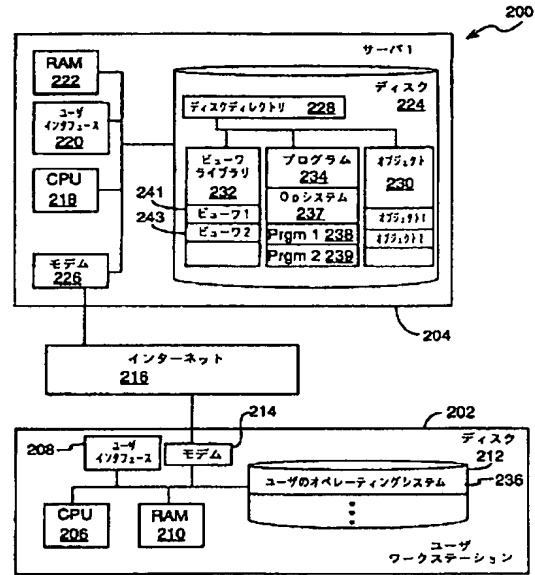
100 従来の分散型コンピュータシステム
 102 第1コンピュータ
 103 コンピュータプログラム
 104 第2コンピュータ
 106 CPU
 108 ユーザインタフェース
 110 主メモリ(RAM)
 112 補助記憶装置
 113 オペレーティングシステム
 114 コミュニケーションインタフェース(モデム) 10
 120 コンピュータネットワーク
 200 従来の分散型コンピュータシステム
 202 第1コンピュータノード(ユーザワークステーションノード)
 204 第2コンピュータノード(サーバノード)
 206 CPU
 208 ユーザインタフェース
 210 主メモリ(RAM)
 212 補助記憶装置(ディスク記憶装置)
 214 コミュニケーションインタフェース(モデム) 20
 216 コンピュータコミュニケーションネットワーク
 218 CPU
 220 ユーザインタフェース
 222 主メモリ(RAM)
 224 補助記憶装置(ディスク記憶装置)
 226 コミュニケーションインタフェース(モデム)
 228 ディスクディレクトリ
 230 オブジェクト
 232 ビューワライブラリ
 234 プログラム 30
 236 オペレーティングシステム
 237 オペレーティングシステム
 238 プログラム
 239 プログラム
 240 バイトコードプログラムペリファイヤ
 241 ビューワ
 242 バイトコードプログラムインタープリタ
 243 ビューワ
 244 バイトコードプログラムコンパイラ
 245 ペリフィケーションステータス 40
 250 本発明による分散型コンピュータシステム
 252 第1コンピュータノード(ユーザワークステーションノード)
 254 第2コンピュータノード(サーバノード)

255 オペレーティングシステム
 256 オペレーティングシステム
 257 CPU
 258 ユーザインタフェース
 260 主メモリ(RAM)
 262 補助記憶装置(ディスク記憶装置)
 264 コミュニケーションインタフェース(モデム)
 266 コンピュータコミュニケーションネットワーク(インターネット)
 267 バイトコードプログラム
 268 CPU
 270 ユーザインタフェース
 272 主メモリ(RAM)
 274 補助記憶装置(ディスク記憶装置)
 276 コミュニケーションインタフェース(モデム)
 280 ディスクディレクトリ
 282 オブジェクト
 283 オブジェクト
 284 ビューワライブラリ
 286 プログラム
 296 クラスローダプログラム
 298 ビューワライブラリ
 302 ペリフィケーションステータス
 324 第2サーバ
 325 補助記憶装置(ディスク記憶装置)
 327 認識されているサーバのリスト
 326 ビューワライブラリ
 340 バイトコードプログラム
 342 スタックカウンタ
 344 仮想スタック
 345 仮想ローカル変数アレイ
 346 スタックスナップショット格納構造
 348 ディレクトリ部
 350 スナップショット部
 352 バイトコード命令
 354 バイトコード命令
 356 バイトコード命令
 358 バイトコード命令
 360 バイトコード命令
 362 バイトコード命令
 364 バイトコード命令
 366 バイトコード命令
 368 バイトコード命令

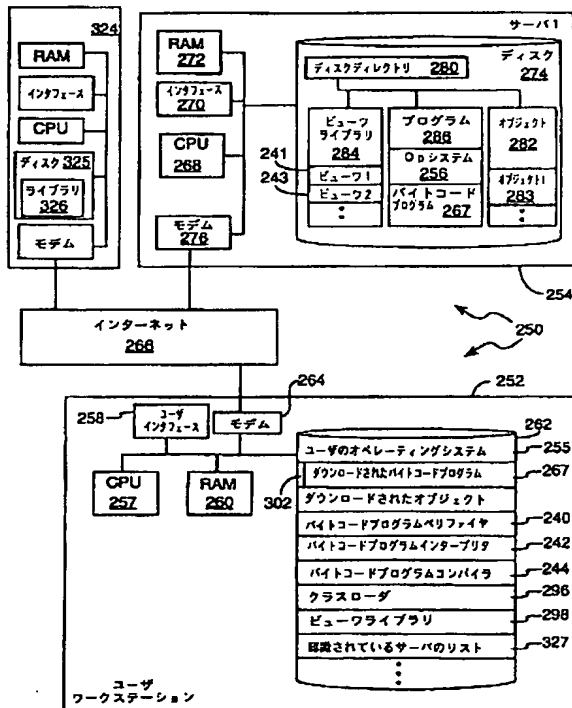
【図 1】



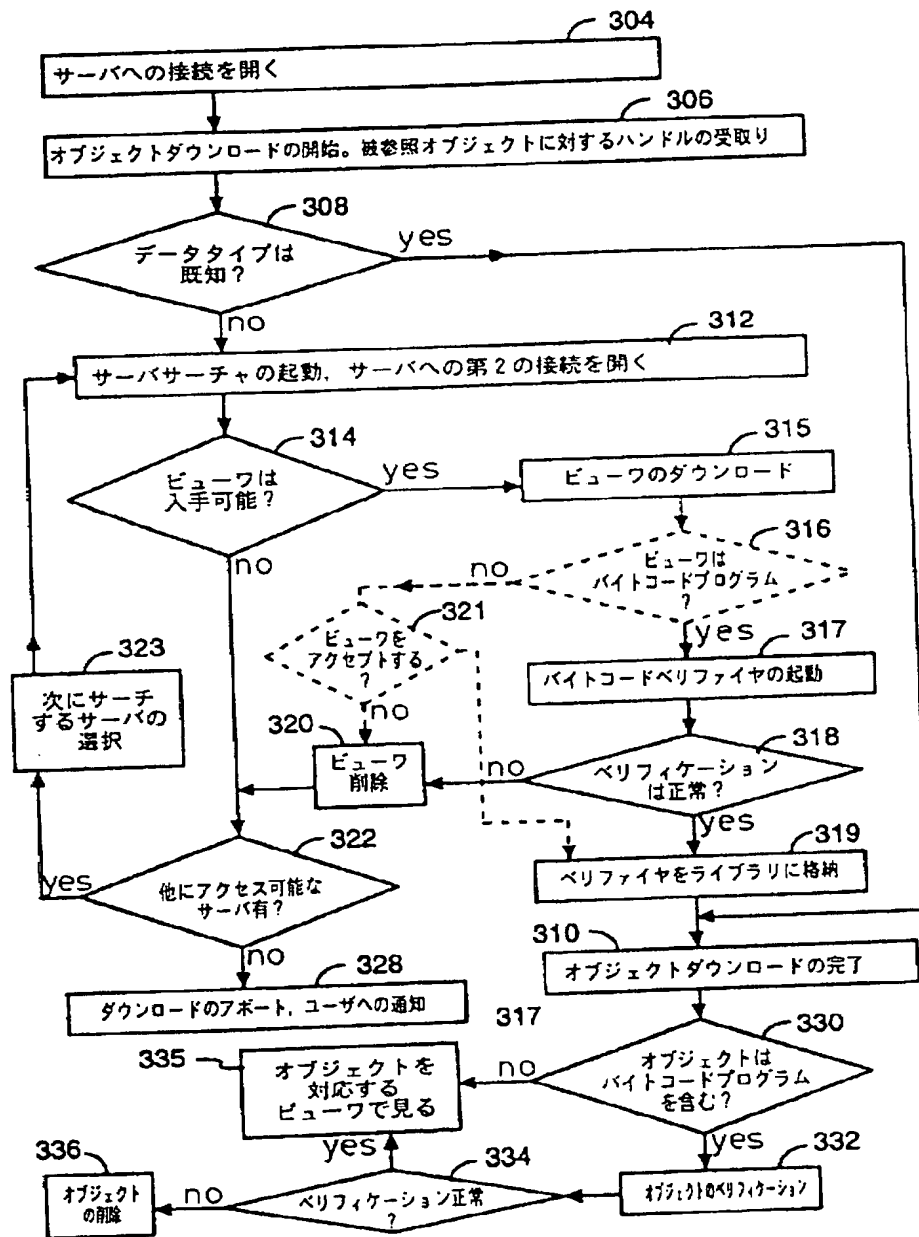
【図 2】



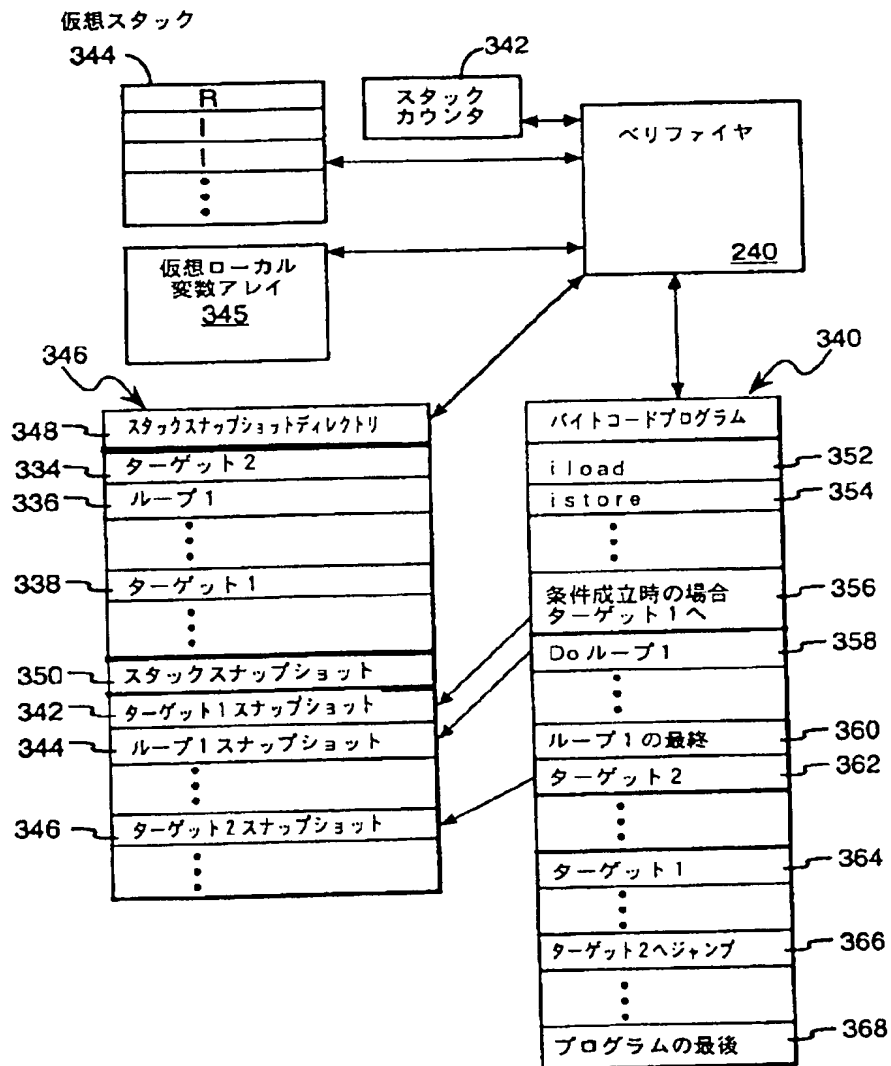
【図 3】



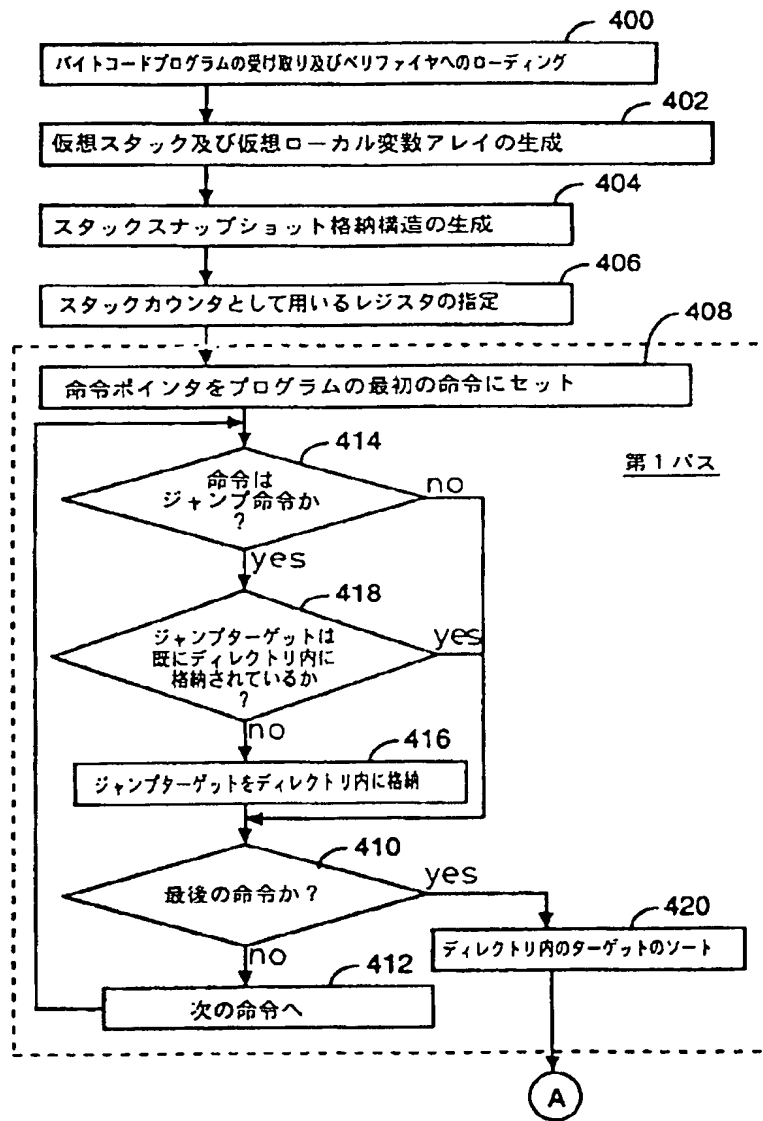
【図4】



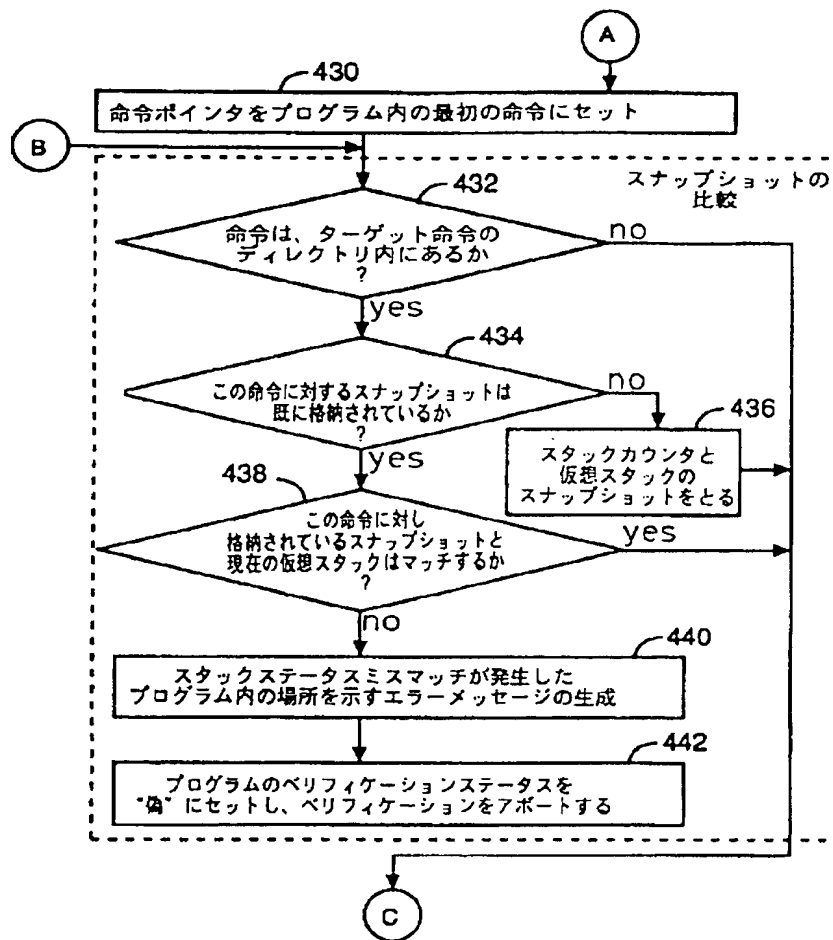
〔図 5〕



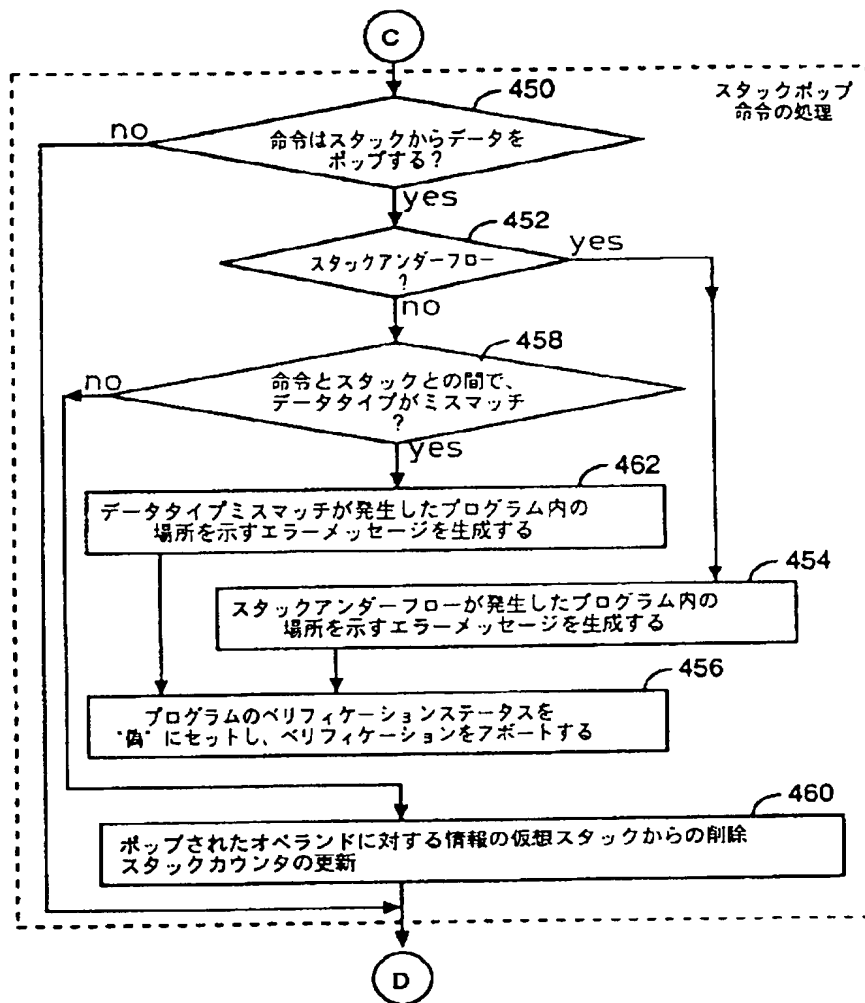
【図 6】



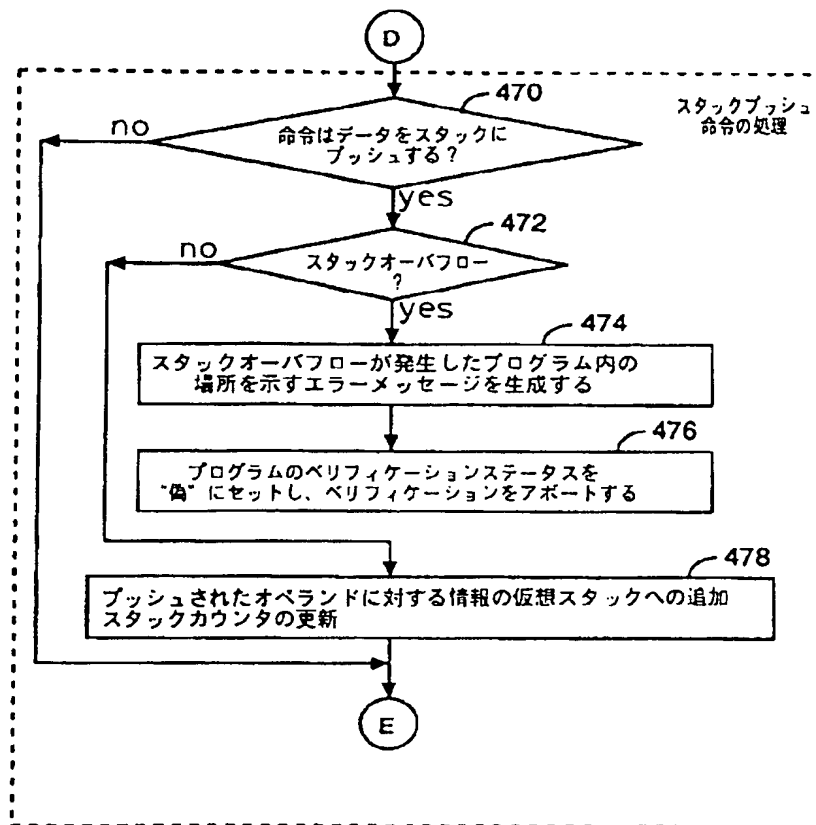
【図 7】



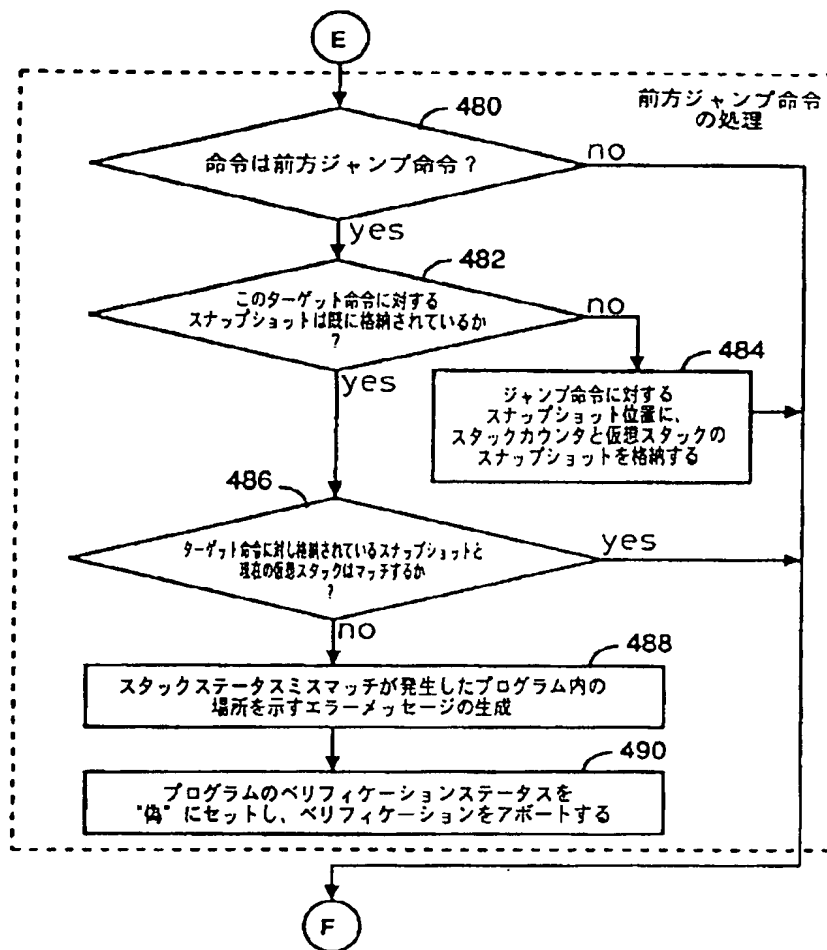
【図 8】



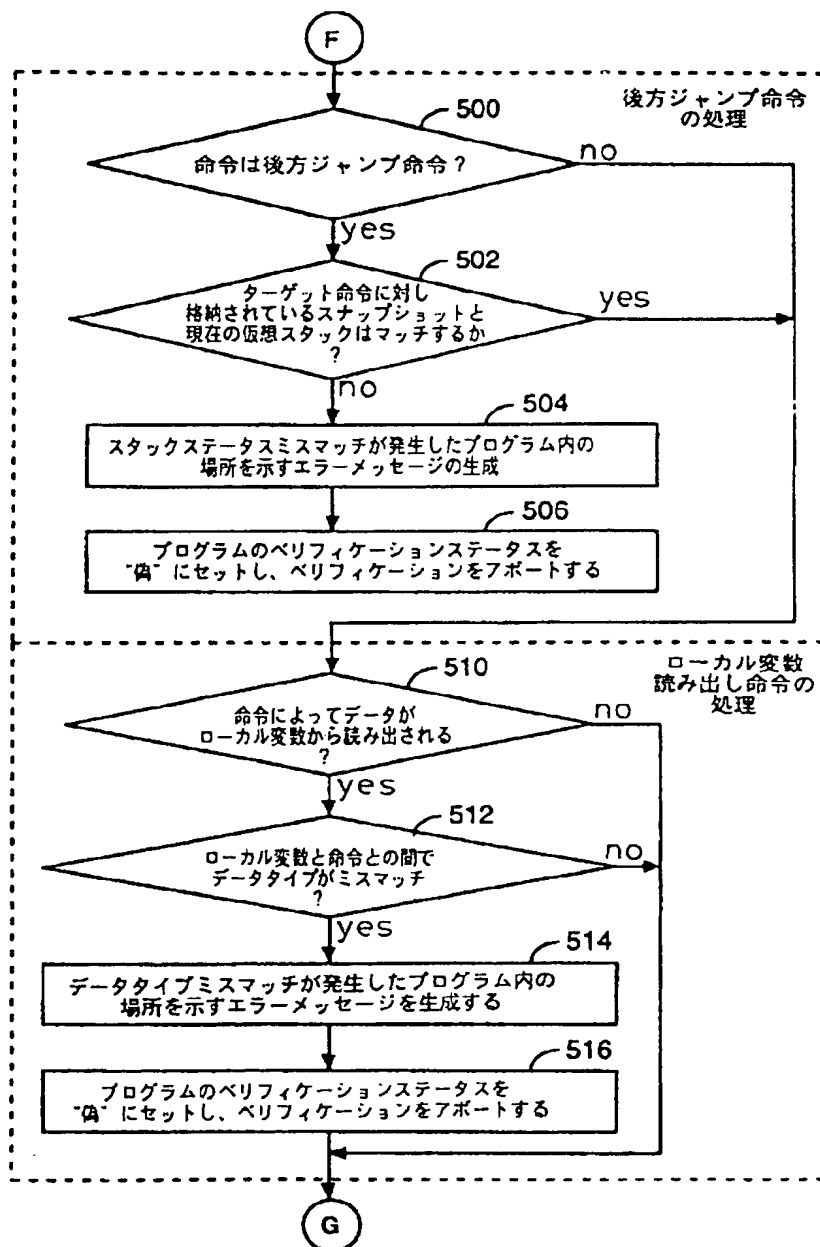
【図 9】



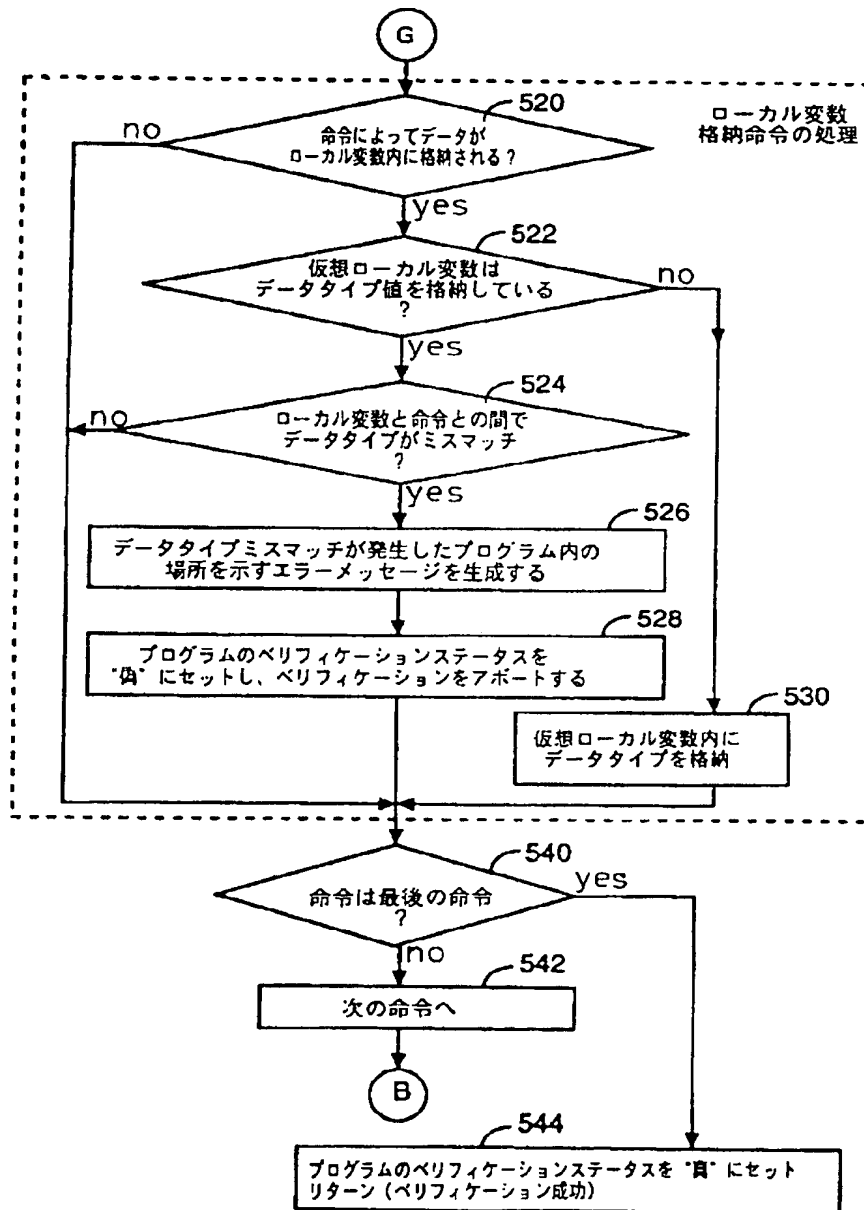
【図 10】



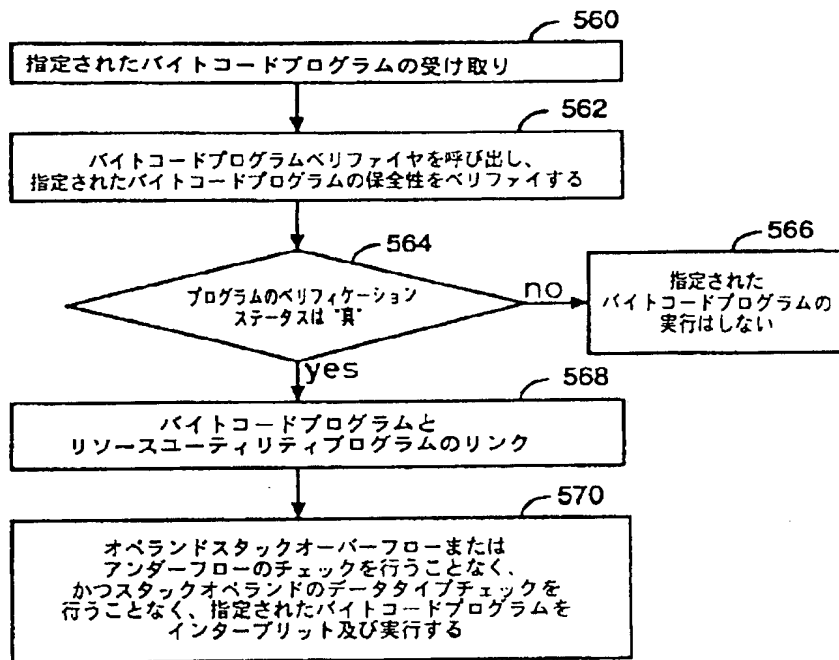
【図 11】



【図12】



【図13】



【公報種別】特許法第 17 条の 2 の規定による補正の掲載
 【部門区分】第 6 部門第 3 区分
 【発行日】平成 15 年 4 月 11 日 (2003. 4. 11)

【公開番号】特開平 8-263447
 【公開日】平成 8 年 10 月 11 日 (1996. 10. 11)
 【年通号数】公開特許公報 8-2635
 【出願番号】特願平 7-349164
 【国際特許分類第 7 版】

G06F 15/16 370
 9/445
 13/00 351

【F I】

G06F 15/16 370 N
 13/00 351 H
 9/06 420 J

【手続補正書】
 【提出日】平成 14 年 12 月 19 日 (2002. 12. 19)
 【手続補正 1】
 【補正対象書類名】明細書
 【補正対象項目名】発明の名称
 【補正方法】変更
 【補正内容】
 【発明の名称】 分散型コンピュータシステムの動作方法及びコンピュータシステムに関連するデバイス及びコンピュータシステムプロダクト
 【手続補正 2】
 【補正対象書類名】明細書
 【補正対象項目名】特許請求の範囲
 【補正方法】変更
 【補正内容】
 【特許請求の範囲】

【請求項 1】 複数の個別のコンピュータを有する分散型コンピュータシステムの動作方法であって、

(a) 第 1 コンピュータに於いて、ビューワプログラムの第 1 ライブラリを格納する過程であって、各ビューワプログラムを以てそのユーザが対応するデータタイプのオブジェクトを見ることが可能にする該過程と、

(b) 前記第 2 コンピュータに於いて、オブジェクト及びビューワプログラムの第 2 ライブラリを格納する過程であって、ビューワプログラムの前記第 1 及び第 2 ライブラリに於ける各ビューワプログラムを以て前記第 2 コンピュータのユーザが対応するデータタイプのオブジェクトを見ることが可能にする該過程と、

(c) 前記第 2 コンピュータに於いて、前記第 2 コンピュータの前記ユーザがオブジェクトを選択することを可能にし、前記選択されたオブジェクトの対応するデータタイプを判定する過程と、

(d) 前記第 2 コンピュータに於いて、前記第 2 コンピュータに格納された前記ビューワプログラムの中に、前記選択されたオブジェクトのデータタイプに対応するビューワプログラムが含まれているかどうか判定する過程と、

(e) 前記過程 (d) での判定結果が否定の場合、前記選択されたオブジェクトのデータタイプに対応するビューワプログラムが前記第 1 コンピュータに格納されているかどうか判定する過程と、

(f) 前記過程 (e) での判定結果が肯定の場合、前記選択されたオブジェクトのデータタイプに対応する前記ビューワプログラムのコピーを前記第 2 コンピュータにロードする過程とを含むことを特徴とする方法。

【請求項 2】 前記過程 (e) での判定結果が否定の場合、前記第 2 データタイプに対応するビューワプログラムが所定の他のコンピュータのセットの何れかの中に格納されているかどうか判定し、前記判定結果が肯定の場合、前記過程 (f) を実行する過程を更に含むことを特徴とする請求項 1 に記載の方法。

【請求項 3】 複数の個別のコンピュータを有する分散型コンピュータシステムの第 1 コンピュータの動作方法であって、

(a) 第 1 コンピュータに於いて、オブジェクトアプリケーションプログラムを格納する過程であって、各オブジェクトアプリケーションプログラムを以てそのユーザが対応するデータタイプのオブジェクトに於けるデータを利用することを可能にする該過程と、

(b) 前記第 1 コンピュータに於いて、前記オブジェクトアプリケーションプログラムの第 1 オブジェクトアプリケーションプログラムを用いて、ユーザが対応する第 1 データタイプを有する第 1 オブジェクトに於けるデータを利用することを可能にする過程であって、前記第 1

オブジェクトの前記データが、第2データタイプの第2オブジェクトを参照するようなリンクデータを含み、前記リンクデータが、前記第2オブジェクトが配置されているような第2コンピュータを識別する該過程と、

(c) 前記第1コンピュータに於いて、前記ユーザが前記リンクデータを選択することを可能にする過程と、

(d) 前記第1コンピュータと前記第2コンピュータとの間に第1通信リンクを確立し、前記第2オブジェクトに対応するデータタイプ情報を検索することを含めて前記第2コンピュータからの前記第2オブジェクトの検索を開始することによって、前記第1コンピュータが前記リンクデータのユーザ選択に回答する過程と、

(e) 前記第1コンピュータに於いて、前記第1コンピュータに格納された前記オブジェクトアプリケーションプログラムが前記第2データタイプに対応するオブジェクトアプリケーションプログラムを含むかどうか判定する過程と、

(f) 前記過程(e)での判定結果が否定の場合、前記第2データタイプに対応するオブジェクトアプリケーションプログラムが前記第2コンピュータに格納されているかどうか判定する過程と、

(g) 前記過程(g)での判定結果が肯定の場合、前記第2データタイプに対応する前記オブジェクトアプリケーションプログラムのコピーを前記第1コンピュータにロードする過程とを含むことを特徴とする方法。

【請求項4】 前記過程(f)に於ける判定結果が否定の場合、前記第2データタイプに対応するオブジェクトアプリケーションプログラムが所定の他のコンピュータのセットの何れかの中に格納されているかどうか判定し、前記判定結果が肯定の場合、前記過程(g)を実行する過程を更に含んでいることを特徴とする請求項3に記載の方法。

【請求項5】 コンピュータシステムと組み合わせて使用するためのコンピュータプログラムプロダクトであって、

該コンピュータプログラムプロダクトが、コンピュータ読取り可能記憶媒体と、そこに組み込まれたコンピュータプログラムメカニズムとを含み、

該コンピュータプログラムメカニズムが、

第2コンピュータに対応するコンピュータメモリに格納された第2オブジェクトへのリファレンスを前記ユーザが選択することを可能にするためのユーザインタフェース制御プログラムと、

第1コンピュータと前記第2コンピュータとの間に第1通信リンクを確立し、前記1つのオブジェクトに対応するデータタイプ情報を検索することを含めて前記第2コンピュータからの前記1つのオブジェクトの検索を開始することによって、オブジェクトリファレンスのユーザ選択に回答するためのコンピュータ間リンク制御プログラムとを含む、

前記ユーザインタフェース制御プログラムが、

前記第1コンピュータに対応するコンピュータメモリに、前記被検索データタイプに対応するオブジェクトアプリケーションプログラムが格納されて存在するかどうか判断し、前記判定結果が否定の場合、前記被検索データタイプに対応するオブジェクトアプリケーションプログラムを前記第2コンピュータに配置しようと試みるための検索命令を含み、

前記コンピュータ間リンク制御プログラムが、

前記検索命令が前記検索されたデータタイプに対応する前記オブジェクトアプリケーションプログラムを前記第2コンピュータに配置する場合、前記第1コンピュータに対応する前記コンピュータメモリに、前記被検索データタイプに対応する前記オブジェクトアプリケーションプログラムのコピーをロードするためのダウンロード命令を含むことを特徴とするコンピュータプログラムプロダクト。

【請求項6】 他のコンピュータと組み合わせて使用するためのコンピュータであって、

ビューワプログラムを格納するためのメモリであって、各ビューワプログラムを以て前記コンピュータのユーザが対応するデータタイプのビューオブジェクトを見ることを可能にする該メモリと、

前記他のコンピュータに格納されたオブジェクトを含めたオブジェクトへのリファレンスをユーザが選択することを可能にするユーザインタフェース制御プログラムと、

前記選択されたオブジェクトリファレンスに於ける情報に従って識別されるような前記他のコンピュータの1つへの第1通信リンクを確立し、前記参照されたオブジェクトに対応する検索データタイプ情報を含めた前記1つのコンピュータからの前記選択されたオブジェクトリファレンスによって参照される前記オブジェクトの検索を開始することによって、前記他のコンピュータに格納された前記オブジェクトの1つを参照するような前記オブジェクトリファレンスの1つのユーザ選択に回答するためのコンピュータ間リンク制御プログラムとを含み、

前記ユーザインタフェース制御プログラムが、

前記コンピュータに格納された前記ビューワプログラムが前記被検索データタイプに対応するビューワプログラムを含むかどうか判定し、前記判定結果が否定の場合、前記被検索データタイプに対応するビューワプログラムを前記1つのコンピュータに配置しようと試みるためのビューワ検索命令を含み、

前記ビューワ検索命令が前記被検索データタイプに対応する前記ビューワプログラムを前記1つのコンピュータに配置する場合、前記被検索データタイプに対応するビューワプログラムのコピーを前記コンピュータの前記メモリにロードするためのダウンロード命令を含むクラスローダを含むことを特徴とするコンピュータ。

【請求項 7】 他のデバイスと組み合わせて使用するためのデバイスであって、

オブジェクトアプリケーションプログラムを格納するためのメモリであって、各オブジェクトアプリケーションプログラムを以って前記デバイスが対応するデータタイプのオブジェクトを利用することを可能にする該メモリと、

前記オブジェクトリファレンスに於ける情報に従って識別されるような前記 1 つのデバイスへの第 1 通信リンクを確立し、前記参照されたオブジェクトに対応するデータタイプ情報の検索を含めて、前記 1 つのコンピュータからの前記オブジェクトリファレンスによって参照される前記オブジェクトの検索を開始することによって、前記他のデバイスの 1 つに格納されたオブジェクトを参照するオブジェクトリファレンスに回答するためのデバイス間リンク制御プログラムと、前記デバイスの前記メモリに格納された前記オブジェクトアプリケーションプログラムが前記被検索データタイプに対応するオブジェクトアプリケーションプログラムを含むかどうか判断し、前記判定結果が否定の場合、前記被検索データタイプに対応するオブジェクトアプリケーションプログラムを前記 1 つのデバイスに配置しようと試みるための検索プログラムと、

前記プログラム検索が前記被検索データタイプに対応する前記オブジェクトアプリケーションプログラムを前記 1 つのコンピュータに配置する場合、前記被検索データタイプに対応する前記オブジェクトアプリケーションプログラムのコピーを前記デバイスの前記メモリにロードするためのダウンロード命令を含むクラスローダとを含むことを特徴とするデバイス。

【請求項 8】 前記デバイス間リンク制御プログラムが、前記被検索データタイプに対応する前記オブジェクトアプリケーションプログラムの前記コピーをダウンロードするための前記 1 つのデバイスへの第 2 通信リンクを確立するための命令を含むことを特徴とする請求項 7 に記載のデバイス。

【請求項 9】 データプロセッサを含むデバイスと組み合わせて使用するためのコンピュータプログラムプロダクトであって、該コンピュータプログラムプロダクトが、コンピュータ読取り可能記憶媒体と、そこに組み込まれたコンピュータプログラムメカニズムとを含み、該コンピュータプログラムメカニズムが、他のデバイスに格納されたオブジェクトを含めたオブジェクトへのリファレンスの選択を可能にする第 1 プログラムと、前記選択されたオブジェクトリファレンスに於ける情報に従って識別されるような前記他のデバイスの 1 つへの第 1 通信リンクを確立し、前記被参照オブジェクトに対

応するデータタイプ情報の検索を含めて、前記 1 つのデバイスからの前記選択されたオブジェクトリファレンスによって参照される前記オブジェクトの検索を開始することによって、前記他のデバイスに格納された前記オブジェクトの 1 つを参照する前記オブジェクトリファレンスの 1 つの選択に回答するためのデバイス間リンク制御プログラムと、

前記デバイスが前記被検索データタイプに対応するオブジェクトアプリケーションプログラムを格納するかどうか判断し、前記判定結果が否定の場合、前記被検索データタイプに対応するオブジェクトアプリケーションプログラムを前記 1 つのデバイスに配置しようと試みるための検索命令と、

前記ビュー検索命令が前記被検索データタイプに対応する前記オブジェクトアプリケーションプログラムを前記 1 つのデバイスに配置する場合、前記被検索データタイプに対応する前記オブジェクトアプリケーションプログラムのコピーを前記デバイスにロードするためのダウンロード命令を含むクラスローダとを含むことを特徴とするコンピュータプログラムプロダクト。

【請求項 10】 前記コピーされたオブジェクトアプリケーションプログラムが、所定のオペランドスタック使用基準を含む所定のプログラムコレクトネス基準に適合するかどうか判定するためのベリフィケーション手順と、

前記ベリフィケーション手順によって前記コピーされたオブジェクトアプリケーションプログラムが前記所定のプログラムコレクトネス基準に適合するかどうか判定する場合、前記ユーザが前記被参照オブジェクトを見ることを可能にするべく、前記コピーされたオブジェクトアプリケーションプログラムの実行を可能にするためのプログラム許可命令とを更に含むことを特徴とする請求項 9 に記載のコンピュータプログラムプロダクト。

【手続補正 3】

【補正対象書類名】明細書

【補正対象項目名】発明の詳細な説明

【補正方法】変更

【補正内容】

【発明の詳細な説明】

【0055】図 6 乃至図 12、及び付録 2 を参照して、バイトコードベリファイヤプログラム 240 の動作について以下に詳細に説明する。付録 2 は、ベリファイヤプログラムを疑似コードで表したものである。付録 2 の疑似コードは、本質的には、汎用コンピュータ言語形式を用いたコンピュータ言語と同等である。この疑似コードは、本発明の説明のためにのみ作られたものであって、本分野の通常の知識を有するコンピュータプログラマであれば容易に理解されるように意図して書かれている。

【手続補正書】

【提出日】平成15年1月14日(2003. 1. 14)

【手続補正3】

【補正対象書類名】明細書

【補正対象項目名】0055

【補正方法】変更

【補正内容】

【0055】図6乃至図12、及び付録2を参照して、

バイトコードベリファイヤプログラム240の動作について以下に詳細に説明する。付録2は、ベリファイヤプログラムを疑似コードで表したものである。付録2の疑似コードは、本質的には、汎用コンピュータ言語形式を用いたコンピュータ言語と同等である。この疑似コードは、本発明の説明のためにのみ作られたものであって、本分野の通常の知識を有するコンピュータプログラマであれば容易に理解されるように意図して書かれている。